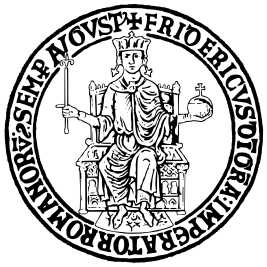
UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"



Scuola Politecnica e delle Scienze di Base
Area Didattica di Scienze Matematiche Fisiche e Naturali

Tesi di Laurea Magistrale in Matematica

Wildfire Spreading: a new application of the Beta distribution

Relatori
Chiar.mo Prof.
Bruno Buonomo
Dott. Gianni Pagnini

Candidata
Benedetta Canfora
Matr. *N*98000618

Anno Accademico 2021/2022

The research presented in this dissertation has been partially funded by BCAM - Basque Center for Applied Mathematics - Bilbao (Spain), where the candidate carried out a 6 months internship in the research line of Statistical Physics.

Contents

Introduction

This dissertation is in the mathematical physics area, more specifically, applications in the statistics field.
The thesis, under the supervision of Dr Gianni Pagnini, was carried out at the BCAM - Basque Centre for Applied Mathematics in Bilbao, Spain. It is the result of the continuous interaction with the team of Statistical Physics, characterised by an international, stimulating and constantly growing environment.

The subject of this thesis is PROPAGATOR: a stochastic cellular automaton model for forest fire spread simulation, conceived as a rapid method for fire risk assessment.
The reason behind the popularity of cellular automata can be traced to their simplicity, and to the enormous potential they hold in modeling complex systems, in spite of their simplicity. Cellular automata can be viewed as a simple model of a spatially extended decentralized system made up of a number of individual components: *cells*. The communication between constituent cells is limited to local interaction.

PROPAGATOR is a cellular automata model which simulates wildfire spread through empirical laws that guarantee probabilistic outputs. This algorithm, whose first version was released in 2009, is currently in use, along with other software, although it is constantly being updated.
In fact, the first version was requested by the Italian Civil Protection, but later it became part of the ANYWHERE project. This project, active from June 2016 to December 2019, was funded under the EU's research and innovation funding program Horizon 2020 (H2020), which aimed to improve emergency management and response to high-impact weather and climate events such as floods, landslides, swells, snowfalls, forest fires, heat waves and droughts.
As part of the ANYWHERE project, Propagator was rewritten in Python. The version we worked with is the 2020 version, but an updated 2022 version

is already available.

The main aim of this work was to understand the distribution of the wildfire propagation. As can be seen from Propagator input parameters, the propagation depends on different factors: ignition point, wind speed and direction, as well as fuel moisture content and firebreaks-fire fighting strategies. Wind is recognized to be by far the most important factor in the entire problem of forest fire propagation. In this paper, we analyzed four different situations varying initial conditions, in particular we changed wind speed: 0 km/h, 10 km/h, 20 km/h, 30 km/h. However, the phenomenon of *fire spotting* and firebreaks-fire fighting strategies were not taken into consideration.

By modifying the code, it was possible to obtain the output required to achieve the desired result. The conclusion we came to is that the distribution of a wildfire spreading is described by the beta distribution.
This allows us, for the first time, to attribute a new application of the beta function: describing the propagation of a process studied using a cellular automaton algorithm.
The thesis is organised as follows:

- In the first chapter, there is an introduction to *special functions*. In particular, their role in applied mathematics is analyzed, followed by a discussion of the two most commonly used special functions: the Gamma function and the Beta function.
  For a more detailed discussion of these topics, we refer to the texts *"Special functions and their applications"* by N. N. Lebedev [18], *"Special Functions, Encyclopaedia of Mathematics and Its Applications"* by G. Andreus, R. Askey and R. Roy [7], from which the thesis has been deduced.

- In the second chapter, the PROPAGATOR model was introduced following the article *"PROPAGATOR: An Operational Cellular-Automata Based Wildfire Simulator"* by A. Trucchia [26].

- The third chapter contains the analysis carried out on the output data. A discussion of the obtained results and suitable observations can be found in the conclusions.

- There are three appendixes containing:

  - Appendix A: the lines of code we wrote to carry out the analysis;

– Appendix B: explanation of the software, apps and routines used, with particular reference to the *Hypathia* server;
– Appendix C: discussion on stochastic processes carried out as an approach and preparation for the subsequent work with Propagator. Key references: *"Stochastic calculus"* di P. Baldi [8] e *"Stochastic processes and applications: diffusion processes, the Fokker-Planck and Langevin equations"* di G. A. Pavliotis [21].

Chapter 1

The Role of Special Functions in Applied Mathematic

In general physics problems are described through partial differential equations which are harder to solve than ordinary differential equations, but the partial differential equations associated with these problems can be reduced to a system of ordinary differential equations through a process known as separation of variables. These ordinary differential equations depend on the choice of coordinate system, which in turn is influenced by the physical configuration of the problem. The solutions of these ordinary differential equations form the majority of the special functions of mathematical physics.

It is the study of differential equations of this kind which lends to the special functions of mathematical physics. The adjective *special* is used in this connection because here we are not, as in analysis, concerned with the general properties of functions, but only with the properties of functions which arise in the solution of special problems [15].
The special functions of mathematical physics arise in the solution of partial differential equations governing the behaviour of certain physical quantities. Probably the most frequently occurring equation of this type in all physics is Laplace's equation

$$\nabla^2\psi = 0 \tag{1.1}$$

satisfied by a certain function ψ describing the physical situation under discussion. The mathematical problem consists of finding those functions which satisfy equation (1.1) and also satisfy certain prescribed conditions on the surfaces bounding the region being considered.

Special function, which include the trigonometric functions, have been used for centuries. Their role in the solution of differential equations as exploited by Newton and Leibniz, and the subject of special functions has been in continuous development ever since. In just the past thirty years several new special functions and applications have been discovered.
They are particular mathematical functions that have more or less established names and notations due to their importance in mathematical analysis, functional analysis, geometry, physics, or other applications [24].
Prominent examples include the gamma function, beta function, hypergeometric function, Whittaker function, and Meijer G-function.

Someone once remarked that special functions would be more appropriately labeled *useful functions*; because of their remarkable properties, special functions have been used for centuries.
In the past years, the discoveries of new special functions and of applications of special functions to new ares of mathematics have initiated a resurgence of interest in this field. About this, we have results from the past but also recent developments.
One reason for the interest in these functions was the fact that several other important functions in mathematics are expressible in terms of hypergeometric functions, special functions represented by the hypergeometric series. They have two significant properties: they satisfy certain identities for special values of the function and they have transformation formulas.

The term is defined by consensus, and thus lacks a general formal definition, but the List of mathematical functions contains functions that are commonly accepted as special, this because in mathematics, some functions or groups of functions are important enough to deserve their own names.
There is a large theory of special functions which developed out of statistics and mathematical physics. A modern, abstract point of view contrasts large function spaces, which are infinite-dimensional and within which most functions are *anonymous*, with special functions picked out by properties such as symmetry, or relationship to harmonic analysis and group representations.

It is common for a special function to be defined in terms of an integral over the range for which that integral converges, but to have its definition extended to a larger domain by analytic continuation in the complex plane or by the establishment of suitable functional relations.
Many special functions appear as solutions of differential equations or integrals of elementary functions. Therefore, tables of integrals usually include descriptions of special functions, and tables of special functions include most important integrals; at least, the integral representation of special functions.

Because symmetries of differential equations are essential to both physics and mathematics, the theory of special functions is closely related to the theory of Lie groups and Lie algebras, as well as certain topics in mathematical physics; in fact the theory of special functions is connected with group representations, with methods of integral representations and with methods in probability theory.

Special functions can be defined by means of power series, generating functions, infinite products, repeated differentiation, integral representations, differential, difference, integral, and functional equations, trigonometric series, or other series in orthogonal functions.
In the broad sense, they could be seen as a set of several classes of functions that arise in the solution of both theoretical and applied problems in various branches of mathematics. In the narrow sense, the special functions of mathematical physics, which arise when solving partial differential equations by the method of separation of variables.

This branch of mathematics has a respectable history with great names, including Gauss, Euler, Fourier, Legendre, Bessel, and Riemann.
All of them spent a lot of time on this topic. A good portion of their work was inspired by physics and the resulting differential equations. These activities culminated in the standard work of Whittaker and Watson about 70 years ago, A Course of Modern Analysis, which has had a great influence and is still important. As well as in applied fields such as electric current, fluid dynamics, heat conduction, wave equation, and quantum mechanics, special functions have extensive applications.
The theory of special functions with its numerous beautiful formulas is very well suited to an algorithmic approach to mathematics.

One of the simplest and most important special functions is the *Gamma function*, knowledge of whose properties is a prerequisite for the study of many other special functions, notably the cylinder functions and the hypergeometric function.
Euler discovered the gamma function, $\Gamma(z)$, when he extended the domain of the function. This function has several representations, but the two most important, found by Euler, represent it as an infinite integral and as a limit of a finite product.
It is illuminating to think of the *Beta function* as a class of integrals that can be evaluated in terms of gamma function.

We refer to gamma and beta functions because, in developing series so-

lutions of differential equations and in other formal calculations, it is often convenient to make use of properties of these two functions [12].

1.1 The Gamma Function

The Gamma function was first introduced to generalize the factorial function to non-integer numbers. The gamma function belongs to the category of the special transcendental functions and its history, reflects many of the major developments within mathematics since the 18th century.
The problem of extending the factorial to non-integer arguments was apparently first considered by Daniel Bernoulli and Christian Goldbach in the 1720s. In particular, in a letter from Bernoulli to Goldbach dated 6 October 1729 Bernoulli introduced the product representation

$$x! = \lim_{n \longrightarrow \infty} (n + 1 + \frac{x}{2})^{x-1} \prod_{k=1}^{n} \frac{k+1}{k+x} \tag{1.2}$$

which is well defined for real values of x other than the negative integers.

Lately Leonard Euler gave two different definitions: the first was an infinite product that is well defined for all complex numbers n other than the negative integers,

$$n! = \prod_{k=1}^{\infty} \frac{(1 + \frac{1}{k})^n}{1 + \frac{n}{k}} \tag{1.3}$$

In the 1729, he wrote about his discovery of the integral representation:

$$n! = \int_0^1 (-lns)^n ds \tag{1.4}$$

which is valid when the real part of the complex number n is strictly greater than -1.

The Gamma function was defined by an infinite product:

$$\Gamma(z) = \frac{1}{z} \sum_{n=1}^{\infty} \frac{(1 + \frac{1}{n})}{(1 + \frac{z}{n})} \tag{1.5}$$

If z be taken as the complex variable $x + iy$, Euler's product for $\Gamma(z)$ converges at every finite z except $z = 0, -1, -2, -3, \ldots$. The function defined by the product is analytic at every finite z except for the singular points just

mentioned. At each of the singular points, $\Gamma(z)$ has a simple pole.
The notation $\Gamma(z)$ and the name *Gamma function* were first used by Legendre in 1814.

From Euler's infinite product for $\Gamma(z)$ can be derived the formula

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt. \tag{1.6}$$

This integral formula is convergent only when the real part of z is positive. Nevertheless this integral formula for $\Gamma(z)$ often is taken as the starting point for introductory treatments of the gamma function. Moreover, the variable z is often confined to real values x, so we define:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt, \ Re\ z > 0 \tag{1.7}$$

whenever the complex variable z has a positive real part Re z.
We can write (1.7) as the sum of two integrals,

$$\Gamma(z) = \int_0^1 t^{z-1} e^{-t} dt + \int_1^\infty t^{z-1} e^{-t} dt, \tag{1.8}$$

where it can easily be shown that the first integral defines a function $P(z)$, which is analytic in the half plane $Re\ z > 0$, while the second integral defines an entire function.
If follows that the function $\Gamma(z) = P(z) + Q(z)$ is analytic in the half-plane $Re\ z > 0$. The values of $\Gamma(z)$ in the rest of the complex plane can be found by analytic continuation of the function defined by (1.7). First, we replace the exponential in the integral for $P(z)$ by its power expansion and we integrate term by term, we obtaining

$$\begin{aligned} P(z) &= \int_0^1 t^{z-1} dt \sum_{k=0}^\infty \frac{(-1)^k}{k!} t^k = \sum_{k=0}^\infty \frac{(-1)^k}{k!} \int_0^1 t^{k+z-1} dt \\ &= \sum_{k=0}^\infty \frac{(-1)^k}{k!} \frac{1}{z+k}, \end{aligned} \tag{1.9}$$

where it is permissible to reverse the order of integration and summation

since

$$\int_0^1 \left|t^{z-1}\right| dt \left|\sum_{k=0}^{\infty} \frac{(-1)^k}{k!} t^k\right| = \int_0^1 t^{x-1} dt \sum_{k=0}^{\infty} \frac{t^k}{k!} = \int_0^1 e^t t^{x-1} dt < \infty \qquad (1.10)$$

The last integral converges for $x = Re\ z > 0$. The terms of the series (1.9) are analytic functions of z, if $z \neq 0, -1, -2, \ldots$. In the region

$$|z + k| \geq \delta > 0,\ k = 0, 1, 2, \ldots,$$

(1.9) is expressed by the convergent series

$$\sum_{k=0}^{\infty} \frac{1}{k!\ \delta}, \qquad (1.11)$$

and, hence, it is uniformly convergent in this region.
We conclude that the sum of the series (1.9) is a meromorphic function with simple poles at the points $z = 0, -1, -2, \ldots$. For $Re\ z > 0$, this function coincides with the integral $P(z)$ and, hence, is the analytic continuation of $P(z)$.
The function $\Gamma(z)$ differs from $P(z)$ by the term $Q(z)$, which is an entire function. Therefore, $\Gamma(z)$ is a meromorphic function of the complex variable z, with simple poles at the points $z = 0, -1, -2, \ldots$.
An analytic expression for $\Gamma(z)$, suitable for defining $\Gamma(z)$ in the whole complex plane, is given by

$$\Gamma(z) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \frac{1}{z + k} + \int_1^{\infty} e^{-t} t^{z-1} dt,\ z \neq 0, -1, -2, \ldots \qquad (1.12)$$

It follows from (1.12) that $\Gamma(z)$ that has the representation

$$\Gamma(z) = \sum_{k=0}^{\infty} \frac{(-1)^n}{n!} \frac{1}{z + n} + \Omega(z + n) \qquad (1.13)$$

in a neighborhood of the pool $z = -n (n = 0, 1, 2 \ldots)$, with regular part $\Omega(z + n)$.

We can define the ***upper*** and ***lower incomplete gamma*** functions that are types of special functions which arise as solutions to various mathematical problems such as certain integrals.

Their respective names stem from their integral definitions, which are

defined similarly to the gamma function but with different or *incomplete* integral limits. As we said, the gamma function is defined as an integral from zero to infinity ((1.6)). This contrasts with the lower incomplete gamma function, which is defined as an integral from zero to a variable upper limit. Similarly, the upper incomplete gamma function is defined as an integral from a variable lower limit to infinity.

The upper incomplete gamma function is defined as:

$$\Gamma(s,x) = \int_x^\infty t^{s-1}\, e^{-t}\, dt, \tag{1.14}$$

whereas the lower incomplete gamma function is defined as:

$$\gamma(s,x) = \int_0^x t^{s-1}\, e^{-t}\, dt. \tag{1.15}$$

In both cases s is a complex parameter, such that the real part of s is positive.

1.1.1 Properties

Euler further discovered some of the gamma function's important functional properties, including the reflection formula.

The gamma function satisfies these three basic relations which play an important role in various transformations and calculations involving $\Gamma(z)$:

- Euler's reflection formula:

$$\Gamma(z)\Gamma(1-z) = \frac{\pi}{sin\pi z}, \quad z \notin \mathbf{Z} \tag{1.16}$$

- Legendre's duplication formula:

$$2^{2z-1}\Gamma(z)\Gamma(z+\frac{1}{2}) = \sqrt{\pi}\Gamma(2z) \tag{1.17}$$

- Stirling's formula:

$$\Gamma(z+1) \sim \sqrt{2\pi z}(\frac{z}{e})^z \tag{1.18}$$

- Raabe's formula:

$$\int_a^{a+1} \ln\Gamma(z)dz = \frac{1}{2}\ln 2\pi + a\ln a - a, \quad a > 0 \tag{1.19}$$

in particular, if $a = 0$ then

$$\int_a^{a+1} ln\Gamma(z)dz = \frac{1}{2}ln2\pi. \tag{1.20}$$

1.1.2 Applications

The gamma function finds application in such diverse areas as quantum physics, astrophysics and fluid dynamics. The gamma distribution, which is formulated in terms of the gamma function, is used in statistics to model a wide range of processes; for example, the time between occurrences of earthquakes.

The primary reason for the gamma function's usefulness in *integration problems*. In such contexts is the prevalence of expressions of the type $f(t)e^{-g(t)}$ which describe processes that decay exponentially in time or space. Integrals of such expressions can occasionally be solved in terms of the gamma function when no elementary solution exists.

Another application can be found in *calculating products*. The gamma function's ability to generalize factorial products immediately leads to applications in many areas of mathematics; in combinatorics, and by extension in areas such as probability theory and the calculation of power series. Many expressions involving products of successive integers can be written as some combination of factorials, the most important example is the binomial coefficient.

An important application of the gamma function is in the *analytic number theory*, the branch of mathematics that studies prime numbers using the tools of mathematical analysis.
In particular, in this field, an application of the gamma function is the study of the Riemann zeta function.

1.2 Beta function

The Beta function was first studied by Euler and Legendre and was given its name by Jacques Binet. Just as the gamma function for integers describes factorials, the beta function can define a binomial coefficient after adjusting indices. The beta function was the first known scattering amplitude in string theory, first conjectured by Gabriele Veneziano. It also occurs in the theory

of the preferential attachment process, a type of stochastic urn process.

The beta function is function of two arguments. As basic definition for the *Beta function* $B(x, y)$ we shall take, as is usually done, the definition

$$B(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt, \tag{1.21}$$

for complex number inputs x, y such that $Re\ x > 0,\ Re\ y > 0$.
By putting $t = \sin^2\theta$ in equation (1.21), we get another definition for the beta function:

$$B(x, y) = 2\int_0^{\frac{\pi}{2}} \sin^{2x-1}\theta \sin^{2y-1}\theta d\theta \tag{1.22}$$

We can define a generalization of the beta function: the ***incomplete beta function***

$$\mathrm{B}(x;\, a, b) = \int_0^x t^{a-1}\,(1-t)^{b-1}\,dt. \tag{1.23}$$

For $x = 1$, the incomplete beta function coincides with the complete beta function. The relationship between the two functions is like that between the gamma function and its generalization the incomplete gamma function.

The regularized incomplete beta function (or regularized beta function for short) is defined in terms of the incomplete beta function and the complete beta function:

$$I_x(a, b) = \frac{\mathrm{B}(x;\, a, b)}{\mathrm{B}(a, b)}. \tag{1.24}$$

The regularized incomplete beta function is the cumulative distribution function of the beta distribution, and is related to the cumulative distribution function of a random variable X from a binomial distribution, where the "probability of success" is p and the sample size is n.

1.2.1 Properties

- Symmetry:

$$B(x, y) = B(y, x) \tag{1.25}$$

Because of the convergent property of definite integrals

$$\int_0^a f(t)dt = \int_0^a f(a-t)dt$$

so we can rewrite the above integral as

$$B(x,y) = \int_0^1 t^{y-1}(1-t)^{x-1}dt,$$

Thus, we get that the beta function is symmetric, $B(x,y) = B(y,x)$.

- Relation with Gamma Function:

$$B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} \tag{1.26}$$

For positive integers x and y, we can define the beta function as

$$B(x,y) = \frac{(x-1)!(y-1)!}{(x+y-1)!}$$

Recall the definition of gamma function (1.6) Now one can write

$$\Gamma(m)\Gamma(n) = \int_0^\infty x^{m-1}e^{-x}dx \int_0^\infty y^{n-1}e^{-y}dy.$$

Then we can rewrite it as a double integral:

$$\Gamma(m)\Gamma(n) = \int_0^\infty \int_0^\infty x^{m-1}y^{n-1}e^{-(x+y)}dxdy.$$

Applying the substitution $x = vtandy = v(1-t)$, we have

$$\Gamma(m)\Gamma(n) = \int_0^1 t^{m-1}(1-t)^{n-1}dt \int_0^\infty v^{m+n-1}e^{-v}dv.$$

Using the definitions of gamma and beta functions, we have

$$\Gamma(m)\Gamma(n) = B(m,n)\Gamma(m+n) \tag{1.27}$$

Hence proved.

- Recurrence Relation:

The recurrence relation of the beta function is given by

$$B(x+1,y) = B(x,y)\frac{x}{x+y}. \tag{1.28}$$

We have

$$B(x+1,y) = \frac{x!(y-1)!}{(x+y)!} = \frac{(x-1)!(y-1)!}{(x+y-1)!} \times \frac{x}{x+y} = B(x,y)\frac{x}{x+y}.$$

From the above relation and because of symmetry of the beta function, the following two results follow immediately:

$$B(x,y+1) = B(x,y)\frac{y}{x+y}$$
$$B(x+1,y) + B(x,y+1) = B(x,y).$$

- Relationship with Central Binomial Coefficient:

$$B(n,n+1) = \frac{1}{n\binom{2n}{n}} \tag{1.29}$$

We observe the reciprocal of central binomial coefficient:

$$\begin{aligned}\frac{1}{\binom{2n}{n}} &= \frac{n!n!}{(2n)!} \\ &= \frac{\Gamma(n+1)\Gamma(n+1)}{\Gamma(2n+1)} \\ &= n\frac{\Gamma(n)\Gamma(n+1)}{\Gamma(2n+1)} \\ &= nB(n,n+1)\end{aligned}$$

This is a really useful relation, especially when solving summations.

1.2.2 Derivatives

The derivative of the beta function is a great way to solve some integrals:

$$\frac{\partial}{\partial x}B(x,y) = B(x,y)\big(\psi(x) - \psi(x+y)\big)$$
$$\frac{\partial^2}{\partial x \partial y}B(x,y) = B(x,y)\big((\psi(x) - \psi(x+y)\big)\big(\psi(y) - \psi(x+y)) - \psi'(x+y)\big). \tag{1.30}$$

where ψ is the *digamma function.*
The digamma function is defined as the logarithmic derivative of the gamma function:

$$\psi(z) = \frac{d}{dz}\ln\Gamma(z) = \frac{\Gamma(z)'}{\Gamma(z)} \sim \ln z - \frac{1}{2z} \tag{1.31}$$

It is the first of the polygamma functions. In mathematics, the polygamma function of order m is a meromorphic function on the complex numbers $\mathbb{C}$, defined as the $(m+1)th$ derivative of the logarithm of the gamma function.

1.2.3 Approximation

Using Stirling's formula, we can easily define the asymptotic approximation of the beta function as

$$\mathrm{B}(x,y) = \frac{(x-1)!(y-1)!}{(x+y-1)!} \sim \sqrt{2\pi}\frac{x^{x-1/2}y^{y-1/2}}{(x+y)^{x+y-1/2}} \tag{1.32}$$

for large x and large y. If on the other hand x is large and y is fixed, then

$$\mathrm{B}(x,y) \sim \Gamma(y)\ x^{-y}. \tag{1.33}$$

1.3 Beta Distribution

Beta distributions are very versatile and a variety of uncertainties can be usefully modeled by them. Many of the finite range distributions encountered in practice can be easily transformed into the standard distribution. It has been applied to model the behavior of random variables limited to intervals of finite length in a wide variety of disciplines.

In probability theory and statistics, the beta distribution is a family of continuous probability distributions defined on the interval $[0,1]$ parameterized by two positive shape parameters, denoted by *alpha* α and *beta* β, that appear as exponents of the random variable and control the shape of the

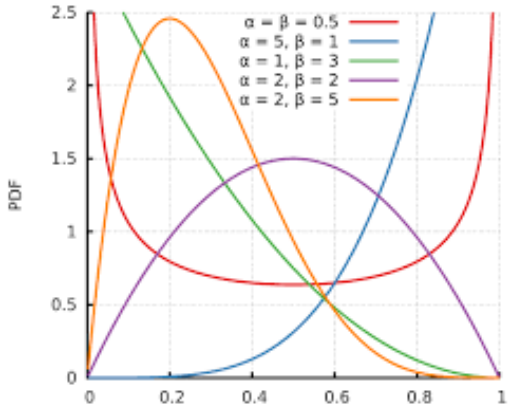


Figure 1.1: Probability density function

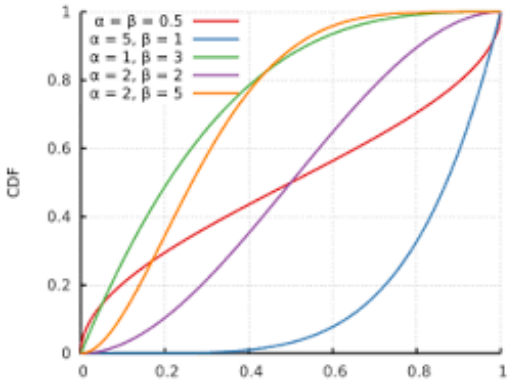


Figure 1.2: Cumulative distribution function

distribution. The generalization to multiple variables is called a *Dirichlet distribution*.

The probability density function (PDF) of the beta distribution, for $0 \leq x \leq 1$, and shape parameters $\alpha, \beta > 0$, is a power function of the variable x and of its reflection $(1 - x)$ as follows:

$$f(x; \alpha, \beta) = \text{constant} \cdot x^{\alpha-1}(1-x)^{\beta-1} \tag{1.34}$$

$$= \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\displaystyle\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\, du} \tag{1.35}$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\, x^{\alpha-1}(1-x)^{\beta-1} \tag{1.36}$$

$$= \frac{1}{\mathrm{B}(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{1.37}$$

where $\Gamma(z)$ is the gamma function. The beta function, B, is a normalization constant to ensure that the total probability is 1. In the above equations x is a realization, an observed value that actually occurred, of a random process X.

The parameters α and β are symmetrically related by

$$f(x; \alpha, \beta) = f(1 - x; \beta, \alpha) \tag{1.38}$$

This implies that if X has the beta distribution with parameters α and β then $1 - X$ has the beta distribution with parameters β and α.
The special case of (1.34) for $\alpha = \beta = 1$ is the uniform distribution. When $\beta = 1$ the beta distribution is known as the power function distribution.
The *cumulative* distribution function of (1.34) is:

$$F(x; \alpha, \beta) = \frac{\mathrm{B}(x; \alpha, \beta)}{\mathrm{B}(\alpha, \beta)} = I_x(\alpha, \beta) \tag{1.39}$$

where $\mathrm{B}(x; \alpha, \beta)$ is the incomplete beta function and $I_x(\alpha, \beta)$ is the regularized incomplete beta function.

1.3.1 Properties

Measures of central tendency

- Mode:
  The mode of a Beta distributed random variable X with $\alpha, \beta > 1$ is the most likely value of the distribution (corresponding to the peak in the PDF), and is given by the expression:

  $$\frac{\alpha - 1}{\alpha + \beta - 2} \tag{1.40}$$

  When both parameters are less than one ($\alpha, \beta < 1$), this is the anti-mode: the lowest point of the probability density curve.
  Letting $\alpha = \beta$, the expression for the mode simplifies to $\frac{1}{2}$, showing that for $\alpha = \beta > 1$ the mode (resp. anti-mode when $\alpha, \beta < 1$), is at the center of the distribution: it is symmetric in those cases.

- Median:
  The median of the beta distribution is the unique real number

  $$x = I^{[-1]}_{\frac{1}{2}}(\alpha, \beta) \tag{1.41}$$

  for which the regularized incomplete beta function $I_x(\alpha, \beta) = \frac{1}{2}$.
  There is no general closed-form expression for the median of the beta distribution for arbitrary values of α and β, but only for particular values of the parameters. For example:

  - for symmetric cases $\alpha = \beta$, median $= \frac{1}{2}$;

- for $\alpha = 1$ and $\beta > 0$, median $= 1 - 2^{-\frac{1}{\beta}}$;
- for $\alpha > 0$ and $\beta = 1$, median $= 2^{-\frac{1}{\alpha}}$

- Mean:
  The expected value (mean) μ of a Beta distribution random variable X with two parameters α and β is a function of only the ratio $\frac{\beta}{\alpha}$ of these parameters:

$$\begin{aligned}\mu = \mathrm{E}[X] &= \int_0^1 x f(x;\alpha,\beta)\, dx \\ &= \int_0^1 x\, \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}\, dx \\ &= \frac{\alpha}{\alpha+\beta} \\ &= \frac{1}{1+\frac{\beta}{\alpha}}\end{aligned}$$

  Letting $\alpha = \beta$ in the above expression one obtains $\mu = \frac{1}{2}$, showing that for $\alpha = \beta$ the mean is at the center of the distribution: it is symmetric.

Measures of statistical dispersion

- Variance:
  The variance (the second moment centered on the mean) of a Beta distribution random variable X with parameters α and β is:

$$\mathrm{var}(X) = \mathrm{E}[(X-\mu)^2] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{1.42}$$

  Letting $\alpha = \beta$ in the above expression one obtains

$$\mathrm{var}(X) = \frac{1}{4(2\beta+1)}, \tag{1.43}$$

  showing that for $\alpha = \beta$ the variance decreases monotonically as $\alpha = \beta$ increases. Setting $\alpha = \beta = 0$ in this expression, one finds the maximum variance $\mathrm{var}(X) = \frac{1}{4}$ which only occurs approaching the limit, $\alpha = \beta = 0$.

  The beta distribution may also be parametrized in terms of its mean

$\mu(0 < \mu < 1)$ and sample size $\nu = \alpha + \beta(\nu > 0)$:

$$\alpha = \mu\nu, \text{ where } \nu = (\alpha + \beta) > 0 \tag{1.44}$$

$$\beta = (1 - \mu)\nu, \text{ where } \nu = (\alpha + \beta) > 0 \tag{1.45}$$

Using this parametrization, one can express the variance in terms of the mean μ and the sample size ν as follows:

$$\text{var}(X) = \frac{\mu(1 - \mu)}{1 + \nu} \tag{1.46}$$

Since $\nu = \alpha + \beta > 0$, it follows that $\text{var}(X) < \mu(1 - \mu)$.
For a symmetric distribution, the mean is at the middle of the distribution, $\mu = \frac{1}{2}$, and therefore:

$$\text{var}(X) = \frac{1}{4(1 + \nu)} \text{ if } \mu = \tfrac{1}{2} \tag{1.47}$$

- Mean absolute deviation around the mean:
  The mean absolute deviation around the mean for the beta distribution with shape parameters α and β is:

$$\text{E}[|X - E[X]|] = \frac{2\alpha^\alpha \beta^\beta}{\text{B}(\alpha, \beta)(\alpha + \beta)^{\alpha+\beta+1}} \tag{1.48}$$

  The mean absolute deviation around the mean is a more robust estimator of statistical dispersion than the standard deviation for beta distributions with tails and inflection points at each side of the mode, Beta(α, β) distributions with $\alpha, \beta > 2$, as it depends on the linear (absolute) deviations rather than the square deviations from the mean. Therefore, the effect of very large deviations from the mean are not as overly weighted.

- Skewness:
  The skewness (the third moment centered on the mean, normalized by the 3/2 power of the variance) of the beta distribution is:

$$\gamma_1 = \frac{\text{E}[(X - \mu)^3]}{(\text{var}(X))^{3/2}} = \frac{2(\beta - \alpha)\sqrt{\alpha + \beta + 1}}{(\alpha + \beta + 2)\sqrt{\alpha\beta}}. \tag{1.49}$$

  Letting $\alpha = \beta$ in the above expression one obtains $\gamma_1 = 0$, showing once again that for $\alpha = \beta$ the distribution is symmetric and hence the skewness is zero. We have positive skew (right-tailed) for $\alpha < \beta$,

negative skew (left-tailed) for $\alpha > \beta$.
It is possible to write the skewness in terms of mean μ and sample size ν:

$$\gamma_1 = \frac{\mathrm{E}[(X-\mu)^3]}{(\mathrm{var}(X))^{3/2}} = \frac{2(1-2\mu)\sqrt{1+\nu}}{(2+\nu)\sqrt{\mu(1-\mu)}}. \tag{1.50}$$

or in terms of variance var and mean μ:

$$\gamma_1 = \frac{\mathrm{E}[(X-\mu)^3]}{(\mathrm{var}(X))^{3/2}} = \frac{2(1-2\mu)\sqrt{\mathrm{var}}}{\mu(1-\mu)+\mathrm{var}} \text{ if } \mathrm{var} < \mu(1-\mu) \tag{1.51}$$

- Kurtosis:
  In probability theory and statistics, kurtosis is a measure of the *tailedness* of the probability distribution of a real-valued random variable. The *excess kurtosis* is defined as kurtosis minus 3.

$$\begin{aligned} excesskurtosis &= kurtosis - 3 \\ &= \frac{E[(X-\mu)^4]}{(var(X)^2)} - 3 \\ &= \frac{6[\alpha^3 - \alpha^2(2\beta-1) + \beta^2(\beta+1) - 2\alpha\beta(\beta+2)]}{\alpha\beta(\alpha+\beta+2)(\alpha+\beta+3} \\ &= \frac{6[(\alpha-\beta)^2(\alpha+\beta+1) - \alpha\beta(\alpha+\beta+2)]}{\alpha\beta(\alpha+\beta+2)(\alpha+\beta+3)}. \end{aligned}$$

Letting $\alpha = \beta$ in the above expression one obtains

$$excesskurtosis = -\frac{6}{3+2\alpha} \; if \; \alpha = \beta \tag{1.52}$$

Therefore, for symmetric beta distributions, the excess kurtosis is negative, increasing from a minimum value of -2 at the limit as $\alpha = \beta \longrightarrow 0$, and approaching a maximum value of zero as $\alpha = \beta \longrightarrow \infty$. The value of -2 is the minimum value of excess kurtosis that any distribution (not just beta distributions, but any distribution of any possible kind) can ever achieve. This minimum value is reached when all the probability density is entirely concentrated at each end $x = 0$ and $x = 1$, with nothing in between.
When rare, extreme values can occur in the beta distribution, the higher its kurtosis; otherwise, the kurtosis is lower. For $\alpha not = \beta$, skewed beta distributions, the excess kurtosis can reach unlimited positive val-

ues (particularly for $\alpha \longrightarrow 0$ for finite β, or for $\beta \longrightarrow 0$ for finite α) because the side away from the mode will produce occasional extreme values. Minimum kurtosis takes place when the mass density is concentrated equally at each end (and therefore the mean is at the center), and there is no probability mass density in between the ends.
Using the parametrization in terms of mean μ and sample size $\nu = \alpha+\beta$:

$$\alpha = \mu\nu, \text{ where } \nu = (\alpha + \beta) > 0 \tag{1.53}$$
$$\beta = (1 - \mu)\nu, \text{ where } \nu = (\alpha + \beta) > 0. \tag{1.54}$$

one can express the excess kurtosis in terms of the mean μ and the sample size ν as follows:

$$excesskurtosis = \frac{6}{3+\nu}\left(\frac{(1-2\mu)^2(1+\nu)}{\mu(1-\mu)(2+\nu)} - 1\right) \tag{1.55}$$

1.4 Applications of the Beta function

Applications of beta function can be found in probability and cumulative density functions. The Beta density function is a very versatile way to represent outcomes like proportions or probabilities.
For example beta function can be used in calculating the probability distributive function of relative wind distributions or modelling the experimental frequency distribution of several meteorological data such as relative sunshine and humidity of a place.
We will discuss some applications here, but for a more complete study, please consult the texts [20], [23], [4], [17].

The Beta function was the first known Scattering amplitude in String theory first conjectured by Gabriele Veneziano, an Italian theoretical physicist and a founder of string theory. Gabriele Veneziano, a research fellow at CERN, in 1968, observed a strange coincidence of many properties of the strong nuclear force that are perfectly described by the Euler beta-function, an obscure formula devised for purely Mathematical reasons two hundred years earlier by Leonhard Euler. In the flurry of research that followed, Yoichiro Nambu of the University of Chicago, Holger Nielsen of the Niels Bohr Institute and Leonard Susskind of Stanford University revealed that the nuclear interactions of elementary particles modelled as one-dimensional strings instead of zero-dimensional particles were described exactly by the Euler beta function. This was, in effect, the birth of string theory. The Euler Beta function appeared in elementary particle physics as a model for

the scattering amplitude in the *dual resonance model.*

A beta distribution is a type of probability distribution that is commonly used to describe uncertainties about the true value of a proportion. There are appropriate distributions to express uncertainties about the prior values for prevalence sensitivity. The beta distribution can be defined by the two parameters, alpha and beta, written as $B(\alpha, \beta)$, with $\alpha = s + 1$ and $\beta = n - s + 1$. Where s is the number of successes out of n trials. If there is no information on which to base a prior distribution, $\alpha = \beta = 1$ should be used. This result is a uniform distribution in which all values between 0 and 1 has equal probability of occurrence. The α and β parameters are best understood in terms of success and failures of an event where $\alpha = success+1$ and $\beta = failures + 1$. Therefore zero success and zero failure may be represented as $B(\alpha = 1, \beta = 1)$ distributions and the mean (expectation) of the beta distribution is $\mu = \frac{1}{1+1} = \frac{1}{2} = 0.5$. Based on zero success and failure the expectation of a success has a 50% probability ($p = 0.5$).
The Beta can be used to describe not only the variety observed across people, but it can also describe your subjective degree of belief (in a Bayesian sense). The Beta distribution can be used to model events which are constrained to take place within an interval defined by a minimum and maximum value. For this reason, the Beta distribution is used extensively in PERT, critical path method (CPM) and other project management or control systems to describe the time of completion of a task.

The beta distribution is a conjugate prior (meaning it has the same functional form therefore also often called convenient prior) to the binomial likelihood function in the Bayesian inferences and as such is often used to describe the uncertainty about a binomial probability.

In *risk analysis*, if we are sure that data is collected according to a binomial process, we can use the beta distribution to describe our uncertainty about the proportion or probability by applying the formula:

$$p = riskbeta(s + 1, n - s + 1)$$

where n is the number of trials of sample and s is the number of successes.

Chapter 2

A Cellular-Automata Fire Simulator: PROPAGATOR

2.1 Mathematical Modeling in Wildfire Management

Mediterranean countries are particularly prone to wildfires, which represent a significant menace to environment, properties, and human lives. Even in countries aware of the fire danger conditions and well equipped for firefighting, there is still a lack of prevention and preparedness capacities in order to deploy in a short time all the activities able to share among the first responders and Civil Protection Authorities (CPAs) the main information to cope with direct impacts on exposed people.

The tragic wildfires that occurred in Greece and in Portugal in the last few years, which caused many fatalities, and the more recent event that occurred in Gran Canaria, where 9000 people were evacuated, constitute examples of the consequences of such shortcomings.
Most of the wild-land fires in the Mediterranean are human caused; however, natural ignition caused by lightning are not negligible and could be increased by climate change. Human-caused fires result from many different reasons, ranging from campfires left unattended to stubble burning, negligently discarded cigarettes, or intentional acts of arson. In addition, wild-land fires can be ignited by anthropic elements such as power lines and railways. Wildfire emergencies, especially in the southern EU Countries, are related with extreme weather conditions, characterized by persistent dry strong winds over flammable land cover species. In this case, the ignition probability increases and, in case it happens, the fire propagation is rapid and difficult to cope

with: in most of the recent wildfire emergencies, casualties happened in few hours after the fire ignition. For this reason, it is extremely urgent to support first responders and CPAs with operational tools in emergency response, based on reliable wildfire risk maps and efficient emergency plans.

The recent dramatic events occurred in Greece and Portugal (such as the mega-fires of June and October 2017) made evident the need of tools able to anticipate the behavior of fire in order to implement prevention and communication activities in time to save lives. This can be achieved using ad hoc mathematical and numerical models, or implementing some other kind of decision-making process.

Unfortunately, physical processes influencing wildfire propagation are complex, meaning that the effects of slopes, wind conditions and fuel moisture interconnect and combine together, determining the evolution of the fire event. Such factors make wildfires multi-scale, multi-physics, and nonlinear phenomena. This makes the formulation of efficient and reliable mathematical models particularly hard, as well as their computational implementation. Nevertheless, in literature, there are many different approaches and models dedicated to this specific task. Such modeling efforts are usually divided into three main approaches [2]:

1. *empirical and semi-empirical models*, which rely on statistically derived laws of fire propagation;

2. *macroscopic-deterministic models*, where the fire spread is modeled in a continuum, mainly by using computational fluid dynamics techniques coupled with atmospheric, heat transfer, and combustion models;

3. *stochastic lattice or grid-based models*, where the evolving quantities are usually described adopting a discretization in space and time, and dealing with the propagation of the fire front from a cell to the neighboring ones by adopting detailed localized evolution rules that comprehend the underlying physics at the desired level of resolution [2],[14].

It is common knowledge that any of the aforementioned modeling is characterized by strengths and drawbacks; in any case the distinction between such categories is not so strict since, in many works, different approaches are mixed together.

To begin with, *empirical models* are quite straightforward to implement and use, and do not require a high computational budget. They have proved

to approximate, under certain restrictions due to their simplified nature, fire-spread dynamics in an acceptable way. However, most such models are derived from controlled laboratory experiments, and their reliability in predicting fires that took place under different conditions and landscapes is to be questioned.
The second category, *macroscopic-deterministic models*, are mostly based on first principles. However, since they try to model intrinsically nonlinear, multi-scale, stochastic, and complex phenomena, their prediction accuracy cannot be always ensured. Moreover, these models typically need high computational resources for simulations on large-scale heterogeneous areas, and-or the computational time is not comparable to the simulated time, making such model not suitable for tactical intervention scenarios.
Grid-based stochastic modeling techniques may thus fill the gap between the first two formulations. Such techniques approximate the complex and inherent stochastic underlying physics, grasping the very mechanisms of fire spread by describing via probabilistic methods the physics at the microscopic/local scale. The front propagation at the macroscopic scale emerges as the result of the rules operating at the detailed (local) level. They are often lightweight models, versatile in the sense that they can be integrated in the framework of existing databases with relative ease.
At the cost of some preliminary modeling, these models can:

- at the physical level incorporate both theoretical/first principles and (semi-) empirical relations inside of the probabilistic mathematical core;

- easily integrate spatial and temporal heterogeneity of the initial and boundary conditions of the simulation, i.e., dealing with spatial patterns of the vegetation type, orography, meteorological conditions.They can easily be coupled with Geographical Information Systems (GIS) and take (possibly real-time) meteorological data as input

Adopting grid-based stochastic modeling techniques, also more complex fire propagation patterns, such as the fire spotting effect, can be also simulated in a rather straightforward way via ad hoc probabilistic rules that may make use of fire intensity, wind field, and fuel characteristics. Cellular Automatas (CA) constitute one of the most well known examples of the latter category of models.

2.1.1 Cellular Automata

From a mathematical point of view, cellular automata (CA) are binary lattices that are updated iteratively. In the automata discussed in this paper, the value of a cell is flipped based only on the number of ones in the neighborhood of the cell to be updated.
We call an automaton *synchronous* if all cells are updated simultaneously, respectively *asynchronous* if the updating affects only one cell at a time. We further call an automaton *deterministic* if the update follows deterministic rules, respectively *probabilistic* if at least one of the following holds:

- the updated cell is picked at random
- the local transition rule is probabilistic - i.e., a cell may flip from zero to one with some probability p, and the same cell may stay in zero with probability $1 - p$.

This work was carried out taking into consideration probabilistic cellular automata.

CA models for wildfire simulation model discretize spatial interactions by adopting a square or hexagonal grid. The macroscopic fire spread dynamics is simulated by the means of an ensemble of different realization of a stochastic process. In every realization, the spreading of the fire front from burning cells to neighboring ones is modeled by the means of probabilistic rules. Although CA models may simplify the underlying physical processes, their modular nature allows them to reach the desired level of complexity and accuracy. This can be achieved under more accurate physical modeling and-or employing state-of-art numerical algorithms [13], [1]. For what concerns the first path, in [9], a CA model has been coupled with existing forest physical models to ensure better accuracy of fire spread simulation. On the other hand, Ghisu et al., in [14], provided elaborate CA models that overcome typical constraints imposed by the shape of the grid and may perform comparably to deterministic models, requiring, however, higher computational budget.

To look at other cellular automata models for forest fire prediction, the reader is referred to [22], [3].

2.2 PROPAGATOR

PROPAGATOR is an operational cellular-automata based software code for simulating forest fires that is already in use, among others, by the European partners of the EU project ANYWHERE H2020, as the Burned Area product within the challenge Weather-induced forest fires, by the Italian Civil Protection, and also by the fire-fighter agency (Vigili del Fuoco) of the Liguria Region (Italy).

Propagator is a tool designed and activated by researchers of the CIMA Research Foundation and it operates at the prototype level since summer 2009. The system is interfaced with the Dewetra platform, which is an integrated system of the Central Functional Center of the Department of National Civil Protection. This platform aims to provide decision-makers with useful information to implement civil protection activities and to support for effective forest fire fighting activities. The system is currently being tested by the Department of National Civil Protection.

Propagator is a model of fire propagation, which is able to simulate the behaviour of a single fire triggered in a given area of the territory. The system is able to provide maps of the probability of fire propagation by predicting the dynamics of the flame front and defining emergency scenarios. In order to do this, propagator is based on the availability of detailed maps of vegetation cover, a digital model of the terrain and the representation of the wind field.

Propagator is a stochastic cellular automaton model for forest fire spread simulation, conceived as a rapid method for fire risk assessment. The model uses high-resolution information such as topography and vegetation cover considering different types of vegetation. Propagator simulates independent realizations of one stochastic fire propagation process, and at each time-step gives as output a map representing the probability of each cell of the domain to be affected by the fire. These probabilities are obtained computing the relative frequency of ignition of each cell. The model capabilities are assessed by reproducing a set of past Mediterranean fires occurred in different countries (Italy and Spain), using when available the real fire fighting patterns.

Propagator simulated such scenarios with affordable computational resources and with short CPU-times. The outputs show a good agreement with the real burned areas, demonstrating that the it can be useful for supporting decisions in Civil Protection and fire management activities.

2.2.1 ANYWHERE project

The first aim of the ANYWHERE (*EnhANcing emergencY management and response to extreme WeatHER and climate Events*) was the development of real-time warning systems for high-impact weather events in order to support emergency response and management.
This project, active from June 2016 to December 2019, was funded under the EU's research and innovation funding program Horizon 2020 (H2020) and had the goal of improving emergency management and response to extreme or high-impact weather and climate events such as floods, landslides, swells, snowfalls, forest fires, heat waves and droughts.

Working in the field of applied research, the project aimed at the implementation of real-time warning systems, able not only to monitor the state of the events but also to identify how many and which areas were at risk. This provided decision makers the opportunity to understand how many people and infrastructure could be involved and thus to obtain a proactive response.

In addition, ANYWHERE contributed to the production of standard tools useful for both decision makers and end users. In fact, the project led to the creation of the A4EU platform, which provides products and services for improving the response to emergencies and high-impact weather and climate phenomena. This platform has been tested in seven pilot sites identified by the project (Liguria; Spain and Catalonia; Canton of Bern, Switzerland; Corsica; Roagaland, Norway; South Savo, Finland).
Public Protection and Disaster Relief (PPDR) institutions have at their disposal a pan-European multi-hazard platform (A4EU) which provides a better identification of the expected weather-induced impacts and their location in time and space before they occur. The operational A4EU prototypes are currently running live at the Civil Protection command centres of seven pilot sites, representing different climatic scenarios around Europe.

Within the project, CIMA Research Foundation was the leader of Work Package 6, concerning the demonstration activity of A4EU in the different pilot sites during the operational period, between October 2018 and September 2019. It should be noted that many of the A4EU systems remained active even at the end of the project, confirming their usefulness.

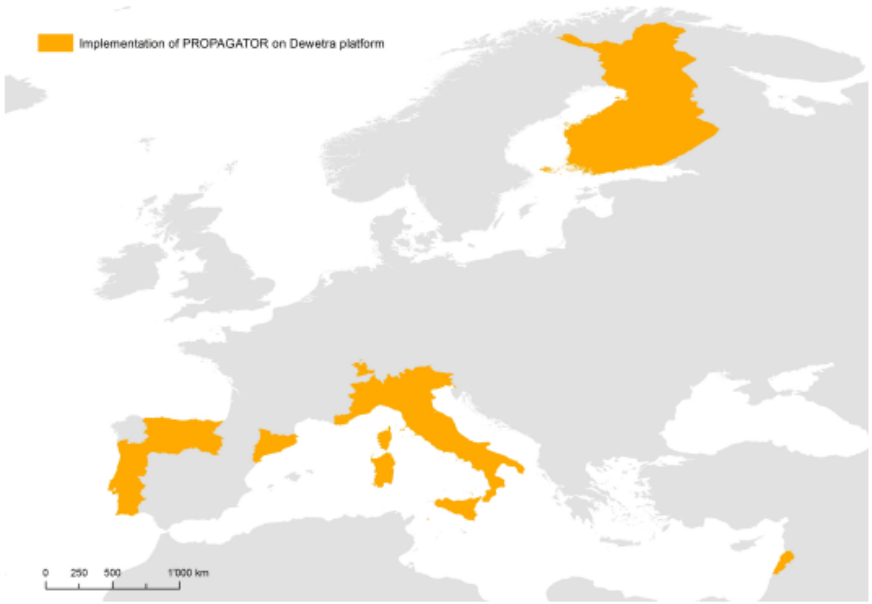


Figure 2.1: All the regions were PROPAGATOR simulations can be run using a Dewetra platform.

2.2.2 History of the development

The implementation of Propagator has spanned across more than a decade; robustness, ease of implementation, and quick operational deployment have always been the beacon during the overall process.
The first implementation of PROPAGATOR started from a request of the Italian Civil Protection Department, to support the organization of the G8 summit 2009, originally planned in La Maddalena, Sardinia, a region frequently affected by severe forest fires in summer season in order to evaluate the best prevention measures and support the fire fighting activities in case of a forest fire event. It was the first time in which it was possible to run fire propagation simulations all over Italy, from a simple web interface. Since its first release, it was able to reproduce burned areas up to 10,000 ha in a few minutes of computational time. Given an ignition point, it highlighted the zones more likely to be affected by fire propagation and the output in this version has been used as a *worst-case scenario*, highlighting the most endangered areas given conditions of totally dry fuel (figure 2.2 letter *a*, *b*).

In 2011 was released the second version of Propagator, in which the model has been implemented in a 3D environment named NAZCA (letter *c* of figure 2.2). The previous version algorithm and server code were mixed: they

were equipped with a Google Web Toolkit based interface, with a server written in MATLAB; at this stage, the algorithm was running as a standalone MATLAB script, much easier to maintain and to develop. The model was shipped as a plugin, together with the data-set of fuel cover and DEM of the whole Italy. This was still quite cumbersome and the object of further improvements. At this stage, taking into account wind and orography, were added timing algorithms, probability maps and isochrones as visual output of the model (letter d of figure 2.2).
At this stage, Propagator did not include a real parametrization of the propagation speed (and thus gave no information on the absolute time scale of the overall process).

In 2014, the third release was completed (letter e of figure 2.2). The web interface was redesigned and it was integrated into the multi purpose MyDewetra platform, a tool for the forecasting, monitoring and real-time surveillance of all the environmental risks.

In 2017, the fourth release saw a total rewriting of the code in the Python programming language, with a new server stack, with Django REST API and database, and Celery task dispatcher (letter f of figure 2.2). Some of the algorithms for the treatment of slope and wind data have been rewritten from scratch, and it has been made possible to change wind conditions over time. An open API had been released to several developers during the ANYWHERE Project (letter g of figure 2.2) and the fuel - DEM dataset had been extended from the sole Italian territory to Finland, Portugal, Spain (Catalonia, Cantabria, Asturias) France (Corsica and Cote D'Azur) and Switzerland. Figure 2.1 portrays the regions in the Europe and Mediterranean basin where Propagator simulations can be run using Dewetra platform and-or the ANYWHERE open API.

In 2020, the fifth release of Propagator (letter h of figure 2.2) saw the implementation of a Rate of Spread (RoS) model in order to give the isochrones a more realistic time parametrization, and the introduction of the fuel moisture into the computational core. The 2020 version also saw the introduction of *fire fighting actions* (lines and polygons where some kind of fire fighting procedure is going to be put in charge) that may be prescribed by the user in a time-dependent way. The Python3 code has been substantially rebuilt according to Object Oriented Programming procedures, in order to rise its modularity for further improvements. In the standalone version, there is the opportunity to furnish directly the data for DEM and fuel cover, in order to launch simulation of any part of the globe, provided that there is available

field data. Running within the Mydewetra platform, meteorological Numerical Weather Prediction (NWP) models have been integrated in Propagator providing synoptic data for wind direction and magnitude. Dead Fine Fuel Moisture Content model data have been integrated as well into the platform.

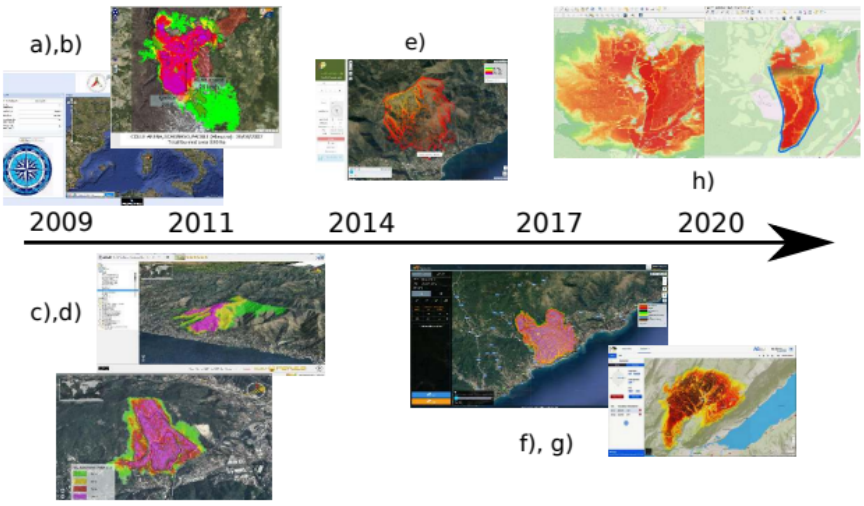


Figure 2.2: The roadmap of operational Propagator implementation, from 2009 to 2020. Each letter represents a milestone in the development of the project.

2.2.3 Propagator: the model

The PROPAGATOR model is a quasi-empirical stochastic cellular automaton model based on a raster implementation and designed for predicting forest wildfires propagation.
The core functioning of the software is based on the discretization of the space into a grid composed of square cells of arbitrary length $\Delta x = \Delta y = L$.

The cell size reflects the resolution in space of the analysis and the final results. Here L has been fixed to 20 m, allowing Propagator to give high resolution output, fundamental for reproducing the middle-sized Mediterranean fires object of the following sections.

Input parameters are wind speed and direction; the ignition point dead fine fuel moisture content and firebreaks-fire fighting strategies can also be considered. The fire spread probability depends on vegetation type, slope,

wind direction and speed, and fuel moisture content instead its speed is determined through the adoption of a Rate of Spread model.
With an appropriate implementation of a ROS model the probability of propagation of the fire from a cell to the adjacent ones for a given time step can be computed. The grid is implemented as a 2D array and for each time step, every cell is characterized by a state taking values from a finite set. More specifically, each cell of the domain can assume one out of three different possible states:

- state 1 corresponds to cells that are *burning* during the current simulation step;
- state 0 corresponds to cells that are *already burned* in previous steps of the simulation;
- state -1 corresponds to cells that are *unburned*, but that can burn in the following steps of the simulation.

At a given time t, an unburned cell has a probability $p^{-1\to 1}$ to burn. Such probability is given by the overall state of the stochastic realization, initial, and boundary conditions. At the subsequent time step of the stochastic process, every burning cell is going to be set to a burned cell (and thus inactive). In operational terms, the computed time step Δt for the state change is appended accordingly to a scheduler which manages the fire propagation mechanism of the cellular automaton. Then after a time step the possible changes in the state of the cells are: an unburned cell can become a burning cell or it can remain unburned, a burning cell becomes a burned cell after one time step with probability equal to 1 and a burned cell can not change its state. The state diagram of this automata is depicted in figure 2.3.

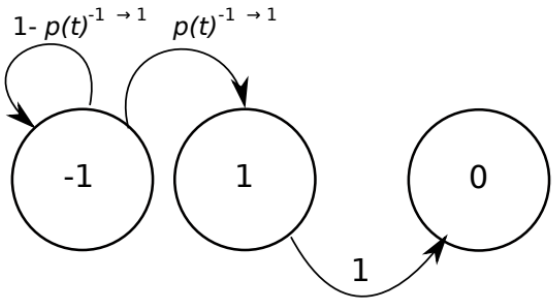


Figure 2.3: The state diagram of the automata adopted in Propagator.

The probability of fire spreading from a cell to one of its neighborhood is p_{ij} and is calculated starting from the nominal fire spread probability p_n, which is then modified considering several factors.
For each cell of the simulation, corresponding to a point $x_P = (x, y)$ of the spatial domain, the model calculates the probability $u(x_P, t)$ of being burnt at time t and space x evaluating the fire frequency for each cell, based on a significant number of stochastic simulations and each simulation is performed for the same ignitions and wind conditions. This procedure is resumed in figure 2.4.

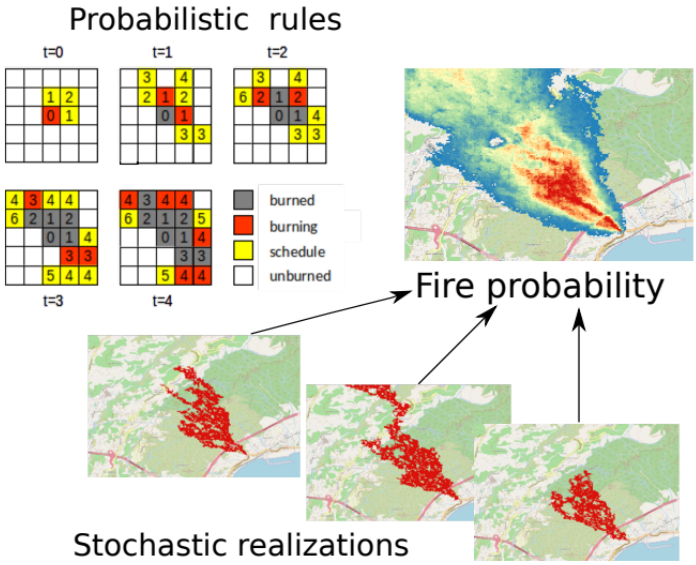


Figure 2.4: Averaging procedure of single realization adopted in Propagator.

The cellular automata is applied on the Moore neighborhood, a two-dimensional square lattice composed of a central cell, the $i - cell$ that is the ignited one, and the eight cells that surround it, as shown in figure 2.5, which can be ignited by the $i - cell$. The fire spreading is stochastically calculated considering the directions between the center of the i-cell and the ones of the neighboring cells, the slopes between the cells and the possible different moisture conditions and considering that each cell is characterized by a vegetation type.

The fire propagates from a cell i to the neighbor cell j with a probability p_{ij}, called Fire Spread Probability, which depends heavily on the involved vegetation types. The p_{ij} is also influenced by the slope between the two

cells, the wind effect (direction and velocity), and the fuel moisture content of the $j - cell$. The probability of the fire propagation p_{ij} from an ignited $i - cell$ at the time t_k to a $j - cell$ is calculated applying the cumulative binomial probability formula:

$$p_{ij} = (1 - (1 - p_n)^{\alpha_{wh}}) \cdot e_m \tag{2.1}$$

where p_n is the nominal Fire Spread Probability, α_{wh} is the factor that combines the topographic and wind influence on the probability and e_m is the factor that simulates the effect of the fine fuel moisture content. In general we can say that an unburned cell has a certain probability of becoming a burning one given that one or more adjacent cells are burning, a burning cell will always become a burned one after a certain time depending on the parameters, and a burned cell remains inactive.

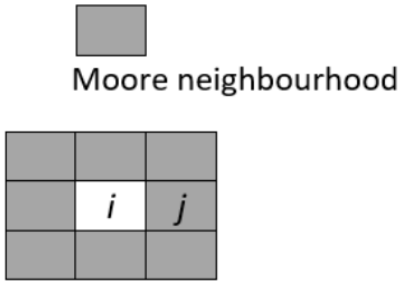


Figure 2.5: Moore neighborhood implemented in Propagator.

The model takes into account the vegetation of the cell that is burning and the cells where the fire can propagate and it analyzes how a certain type of vegetation can ignite other types of vegetation, or also the same vegetation type. These probability values are given in input through a fire spread probability table, table in figure 2.6, which considers all the possible combinations between the different vegetation and land-cover types.
In particular, in order to preserve ease of use and portability, Propagator adopts a manageable simplified custom fuel model with *seven* available fuel types corresponding to *seven* different types of vegetation. The considered fuel types are the following: broad-leaves, shrubs, grasslands, fire-prone conifers, agro-forestry areas, non-fire prone forest, and non-vegetated areas. The nominal Fire Spread Probability p_n represents the possibility for the $i - cell$, characterized by a certain vegetation cover, to ignite an adjacent $j - cell$, characterized by the same, or another, vegetation cover. The class

called *non-vegetated areas* includes man-made buildings and infrastructures (e.g., streets, villages and towns) and the non-vegetated terrains, such as natural bare soil (rivers, lakes, and seas are considered by default as non-burnable areas as well). Fire propagation cannot take place in this class.

		Burning Cell					
		Broadleaves	**Shrubs**	**Grassland**	**Fire-Prone Conifers**	**Agro-Forestry Areas**	**Not Fire-Prone Forest**
neighbor cells	Broadleaves	0.3	0.375	0.25	0.275	0.25	0.25
	Shrubs	0.375	0.375	0.35	0.4	0.3	0.375
	Grassland	0.45	0.475	0.475	0.475	0.375	0.475
	Fire-prone conifers	0.225	0.325	0.25	0.35	0.2	0.35
	Agro-forestry areas	0.25	0.25	0.3	0.475	0.35	0.25
	Not fire-prone forest	0.075	0.1	0.075	0.275	0.075	0.075
Nominal Fire Spread Velocity [m/min]		100	140	120	200	120	60

Figure 2.6: n the first six rows, the values of the nominal fire spread probability p_n between all the species are given. In the last row, nominal fire spread velocity v_n is reported.

The p_n values were defined according to a continuous and thorough calibration through all the development of the model, and valuable information deriving from fire susceptibility mapping.
The slope and the wind speed and direction can modify the initial value of p_n, increasing or decreasing the nominal value depending on the direction of propagation. The influence of the topography is taken into account through the slope between the two cells. The slope increases the propagation probability p_n when the slope increases in the direction of propagation (uphill case) and it decreases p_n if slope decreases in the direction of propagation (downhill case). In general, macroscale factors, such as the atmospheric stability, but also mesoscale factors, have a big influence on the propagation of the wildfire [10]. For a more in-depth look at the phenomenology of the phenomenon, see [5], [6], [27].

2.3 The role of the wind in wildfire propagation

A model for predicting a wildfire spread would have to take into account external environmental factors like meteorological conditions as well as specific characteristics of the terrain. The most important factors that affect the rate of spread and shape of a forest fire front are the fuel type (type of vegetation) and humidity, wind speed and direction, forest topography (slope and

natural barriers), fuel continuity (vegetation thickness).

Many empirical relationships have been suggested in the literature to model the effect of wind on the rate of fire spread. Here the wind influence is taken into account by considering the wind velocity and its direction (both considered to be homogeneous in the domain) relative to the direction from the burning cell and the adjacent ones. At every time step, the value of wind for every cell is perturbed in both magnitude and direction.
The influence is significant only if wind speed is quite high: in the low-speed case, it does not modify the probability of propagation, not increasing nor decreasing. However, when the wind speed is sufficiently high, it has a big impact on the probability of propagation. The wind direction plays a key role in the overall process because the probability of burning is increased when the propagation direction is aligned with the wind direction, while it is decreased when the directions are opposite. The wind influencing factor α_w is shown in figure 2.7.

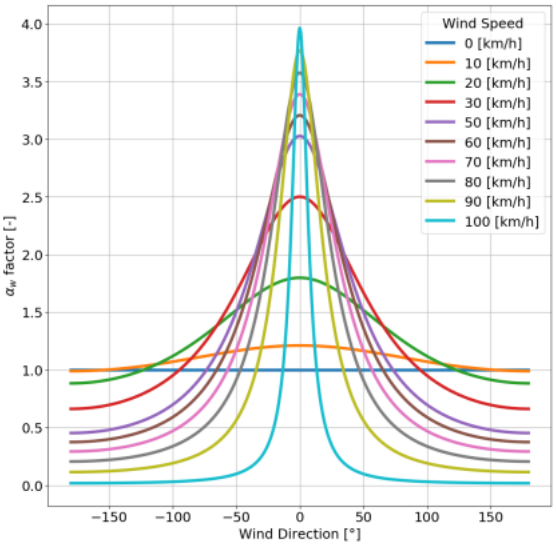


Figure 2.7: Wind influencing factor α_w

The factor that combines the topographic influence α_h with the wind influence α_w on the Fire Spread Probability, named α_{wh}, is obtained as follows:

$$\alpha_{wh} = \alpha_w \cdot \alpha_h \tag{2.2}$$

The way α_{wh} impacts the Fire Spread Probability, p is shown in figure 2.8. Ii is possible to see that: when α_{wh} is equal to 1, the nominal80 probability is

obtained; when this factor is not unitary, it is possible to evaluate the effect of the possible combinations of slope, wind speed, and direction. The Fire Spread Probability can thus vary in a range between at about zero and 0.7, a value that in numerical experiments makes propagation quite always possible.

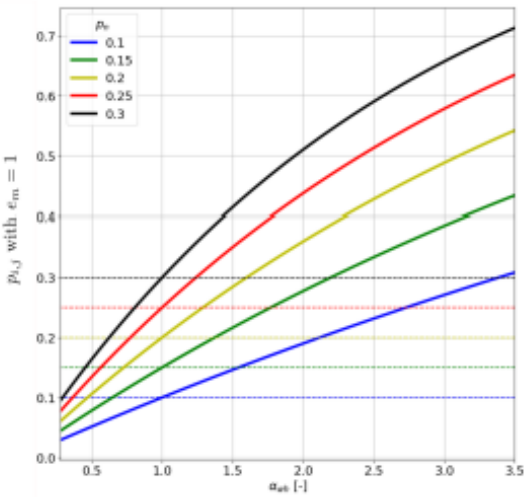


Figure 2.8: Influence of the combined slope-wind factor on the Fire Spread Probability. The plot portrays the dependence of p_{ij} of formula (2.1) on the slope-wind factor α_{wh}, given a fixed $e_m = 1$ and several values for p_n.

When a cell is ignited, the transition time of the fire is modeled by combining the Rate of Spread v_{prop} and the fuel moisture factor f_m, with the distance d from the center of the $i - cell$ which propagated the fire to the center of the newly ignited $j - cell$. In particular, the transition time Δt is calculated as:

$$\Delta t = \frac{d}{v_{prop} \cdot f_m} \tag{2.3}$$

It is acknowledged that the flammability of the vegetal fuel and, consequently, the rate of the spread of a fire depends exponentially on the fuel moisture content. f_m is calculated using the following formulation:

$$f_m = e^{c\dot{M}_n} \tag{2.4}$$

where c is a constant that has been set at -0.014 and M_n is the fuel moisture (ranging from 0 to 1).
The Rate of Spread v_{prop} is calculated starting from the nominal v_n, which stands for the Fire Spread Velocity for each vegetation type without slope

and wind effects, and then modifying it by considering the slope and the wind effects. Values for v_n are reported in the table in figure 2.6 . Slope and wind effects have been evaluated through the formulations calculated as:

- the wind speed factor K_w is evaluated as

$$K_w = exp(0.1783V) \tag{2.5}$$

  where V is the wind velocity in the direction of propagation, in [m/s];

- the slope factor K_Φ is evaluated as

$$K_\Phi = exp(3.533(tan\Phi)^{1.2}) \tag{2.6}$$

  where Φ is the terrain slope angle in the direction of propagation.

The Rate of Spread v_{prop} is then evaluated multiplying the nominal Fire Spread Velocity v_n by the two factors, K_w and K_Φ.

Fire spotting is an important phenomenon associated with wildfires. It is documented as a dominant aspect that has contributed to the rampant spread of fire in many devastating historical fires. Spot fires occur when fragments of the fuel tear off from the main fuel source and horizontal wind transports the burning embers beyond the zone of direct ignition. The burning embers/firebrands can develop new secondary ignition spots and lead to a perilous increase in the effective rate of spread of the fire [25].

The fire-spotting mechanism over varying wind speeds shows similar behaviour for the different sized firebrands. The effective burnt area increases with increasing wind speeds, but after a certain threshold an increase in the wind speed leads to a decline in the effective burnt area. Wind speed affect the ability of spot fires to accelerate wildfire propagation [19], [11].

At the same time, the increasing wind speed also causes a decrease in the magnitude of the probability; therefore, beyond a certain threshold the overall contribution from fire spotting starts to fall.
In terms of the physical quantities used in the parameterisation, it can be argued that strong winds can carry the firebrands longer distances from the main source and result in a larger fire perimeter (increasing "long-range probability").

Historically it has been reported that strong winds coupled with extremely dry conditions formed the perfect recipe for long-range fire spotting. Strong

wind speeds can loft the smaller firebrands longer distances, but with an increasing wind speed the combustion process quickens and the firebrands reach the ground at a lower temperature. This fact explains the reduced effect of fire spotting on the burned area at high wind conditions.

Chapter 3

Analysis and Results

In this chapter we will show all the analysis we did and all the results we obtained.
Propagator input parameters are:

- ignition point: 44.4099;9.1774
- wind speed: floating
- wind direction: 0
- time limit: 1000 min
- time resolution: 10 min
- number of threads: 1000
- dimension of the grid: 20 km
- moisture: 2

We consider *uniform* vegetation and *uniform* Digital Elevation Model (DEM), we are also considering no fire spotting and no fire fighting actions.

To better study the wildfire propagation we simulated three different situations changing the value of wind speed, we considered wind speed equal to: $0km/h$, $10km/h$, $20km/h$, $30km/h$.
The aim is to understand how is better to describe the fire propagation and we will see that the best approximation is from the *beta distribution*.

Propagator outputs:

- $A[11 \times 2]$: a matrix with the mean area values every 1000 realizations in the first column and the corresponding instant of time in the second one (this is like a report of the situation every 10 minutes);
- $X[11011 \times 2]$: a matrix with the random area values for each realization in the first column and the corresponding instant of time in the second one.

We obtained the matrix X with a modification of the code because the default output of propagator was only the matrix A.

3.1 Steps of the analysis

The first thing we did was a statistical analysis which can be divided in the following sections:

- **Data Separation:** given the output X matrix, we split it in $[1000 \times 2]$ matrices. Each of them has in the first column the area value of each tread for a particular instant of time and in second one the referred instant of time;
- **Data Cleansing:** given the X matrix of output, we remove the area values that don't vary with time. From a phenomenological point of view we delete the cells which do not contribute to the extension of the fire;
- **Histogram representation:** for each time step we construct the corresponding histogram considering an empirical distribution. In order to appreciate the trend over the time, we plot all the histograms on the same graph;
- **Study of trends:** we look at the trend of mean area, variance, correlation, pdf maximum, last point;
- **Data Scaling:** we scale the data of X matrix and they are indicated with Y;
- **Parameters calculation:** with the help of a python function, it has been possible to calculate, in an empirical way, the parameters required for the beta distribution;
- **Data fitting:** thanks to the calculated parameters a fit of the data is performed;

- **Analysis of moments of various order:** we compare mean, variance, skewness and kurtosis of the beta fit with that of the data from which it is obtained.

The described analysis was conducted for the four different cases which will be shown in detail below. All the results discussed here, derive from the codes in the Appendix A.

3.2 Case I: no wind

- **Data separation and data cleansing:** Given the output matrix X, it has been split in the different matrices. The matrix $X1$ doesn't appear because at the time $t = 10min$ there is no new burnt cell.

X0[1000]	t=0	X6[949]	t=60
X2[969]	t=20	X7[949]	t=70
X3[951]	t=30	X8[949]	t=80
X4[949]	t=40	X9[949]	t=90
X5[949]	t=50	X10[949]	t=100

Table 3.1: The X matrix is divided into column vectors representing the number of threads for each time t.

- **Histogram representation:** Shown below the histograms for each time step with the correspondent empirical distribution.

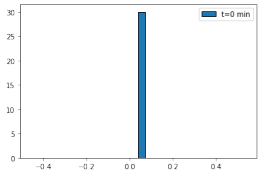


Figure 3.1: $X0[1000],\ t = 0.$

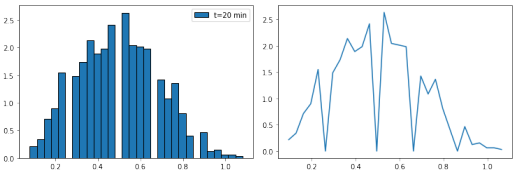


Figure 3.2: $X2[969],\ t = 20.$

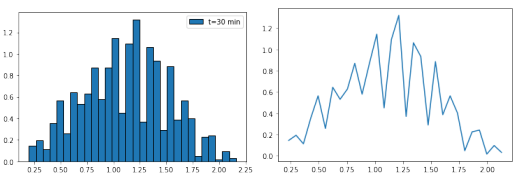


Figure 3.3: $X3[951],\ t = 30.$

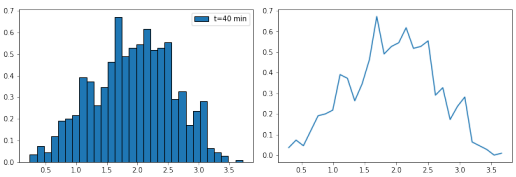


Figure 3.4: $X4[949],\ t = 40.$

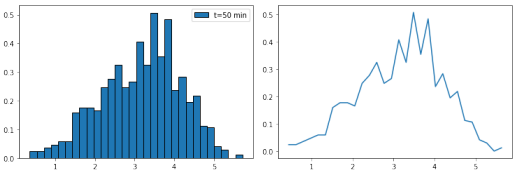


Figure 3.5: $X5[949],\ t = 50.$

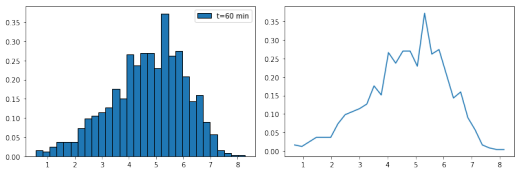


Figure 3.6: $X6[949],\ t = 60.$

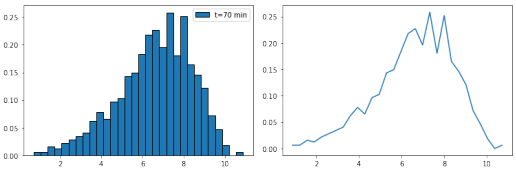


Figure 3.7: $X7[949],\ t = 70.$

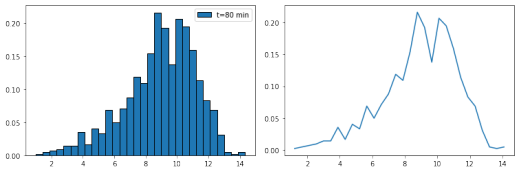


Figure 3.8: $X8[949],\ t = 40.$

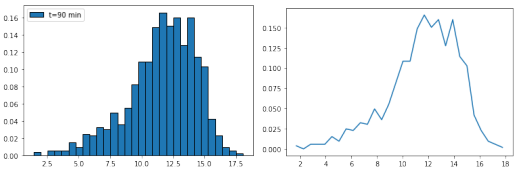


Figure 3.9: $X9[949],\ t = 90.$

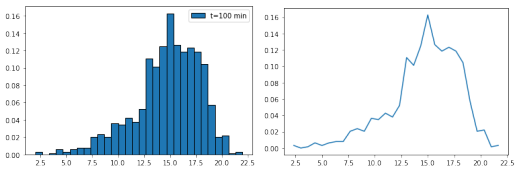


Figure 3.10: $X10[949],\ t = 100.$

The histograms and the empirical distributions obtained for each individual vector are plotted on the same graph to observe its trend over time:

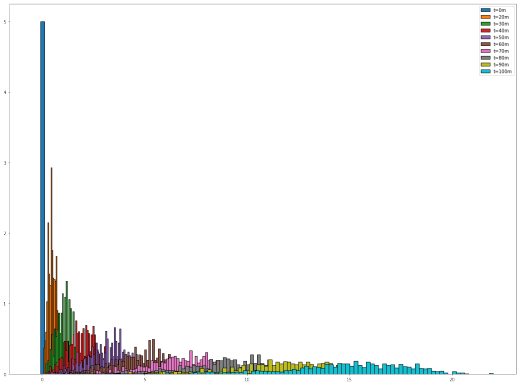


Figure 3.11: Histograms trend over the time

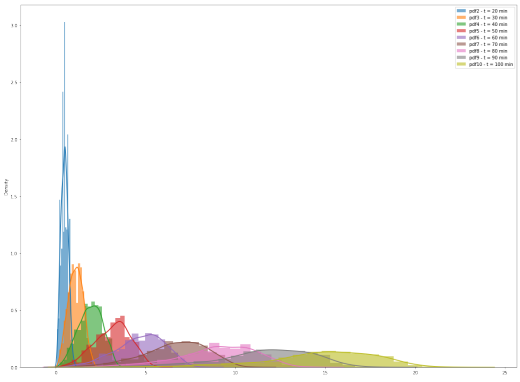


Figure 3.12: Distribution trend over the time

Considering the variable $X - A$, i.e. area for each tread minus A mean, a change in the trend and in the accumulation point can be observed:

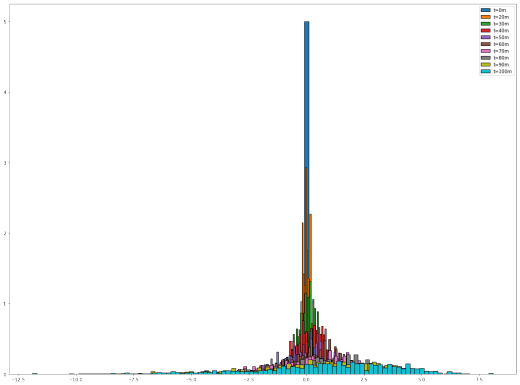


Figure 3.13: Histograms trend over the time

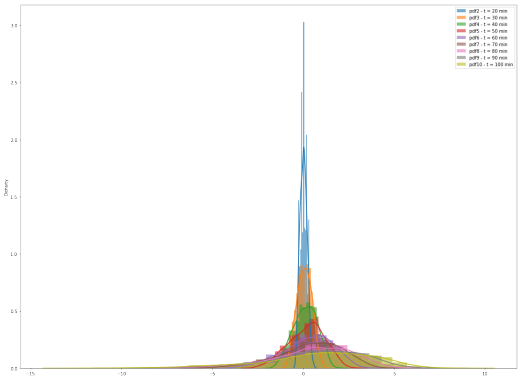


Figure 3.14: Distribution trend over the time

- **Study of trends:**

  Considering the outputs, we study the trends of mean, variance, correlation, maximum of the distribution and last point. Note that maximum of the distribution means the element belonging to the dataset to which the highest point of the curve corresponds.

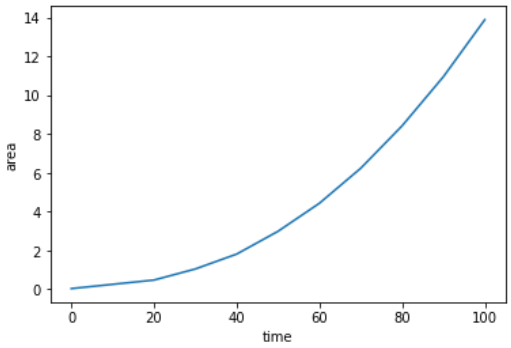


Figure 3.15: Mean Area trend

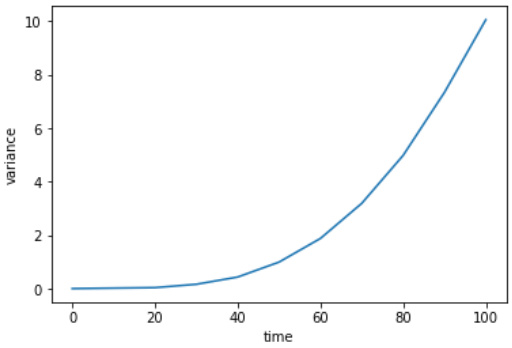


Figure 3.16: Variance trend

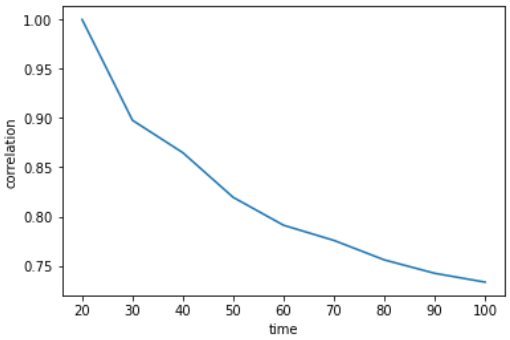


Figure 3.17: Correlation trend

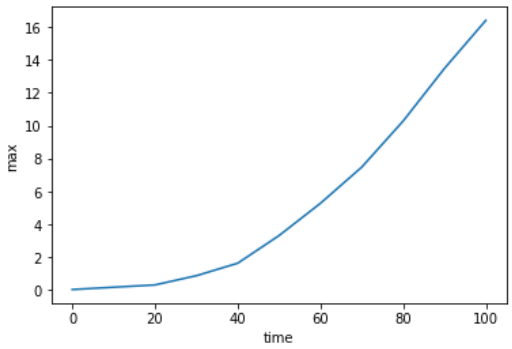


Figure 3.18: Trend of the maximum of the pdf

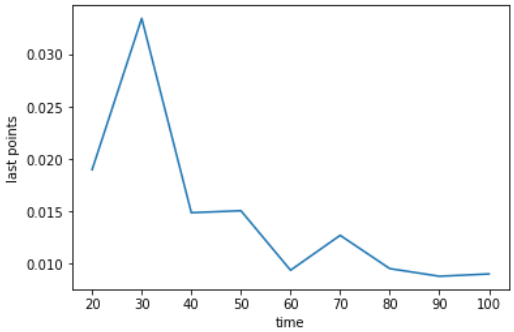


Figure 3.19: Last point trend

- **Data scaling:** To proceed with the analysis we now consider the scaled data, therefore we calculate

$$Y = \frac{X - A}{\sqrt{var(X)}} \tag{3.1}$$

as well as the X matrix, the Y matrix is divided into column vectors corresponding to the instants of observation.
We conducted the same analysis reported above also for the Y vectors but in order to focus the attention on the successive results, we do not show it.

- **Parameters calculation and data fitting:** Considering the variable Y, our goal is to find the parameters for it to be a beta distribution. For this purpose, we can use the *beta.fit* Python function (Appendix B) which calculate, in an empirically way, the parameters α and β.
In order to use *beta.fit* it is necessary a data scaling in the interval $[0, 1]$.
Using the obtained values of the needed parameters, a fit of the data is performed obtaining the following plots:

Figure 3.20: Linear beta fit

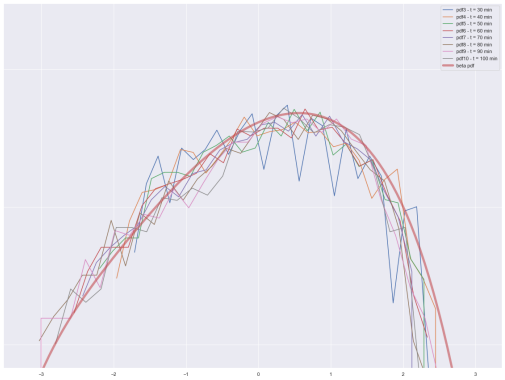


Figure 3.21: Logarithmic beta fit

- **Analysis of moments of various order:** Considering the four moments (mean, variance, skewness and kurtosis), they are calculated on the scaled Y vectors and their distribution. The results are shown in the following plots:

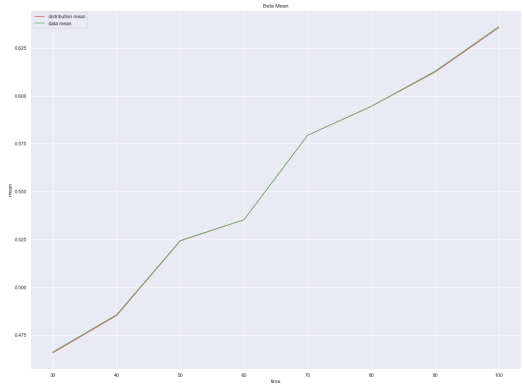


Figure 3.22: Mean

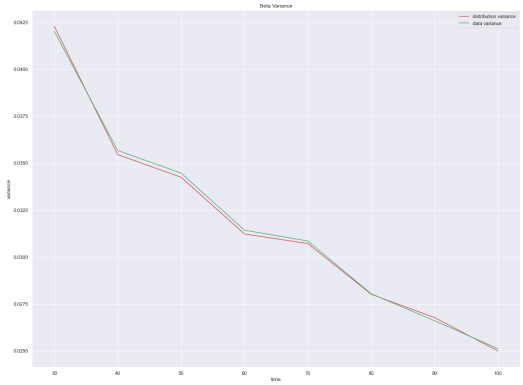


Figure 3.23: Variance

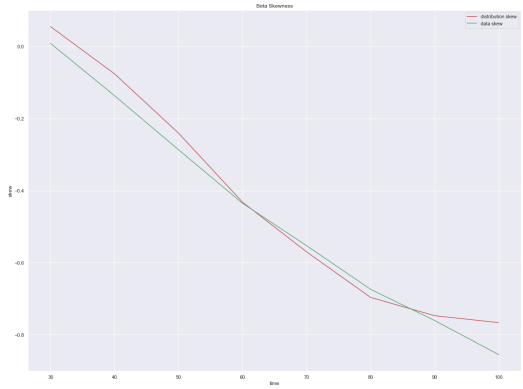


Figure 3.24: Skewness

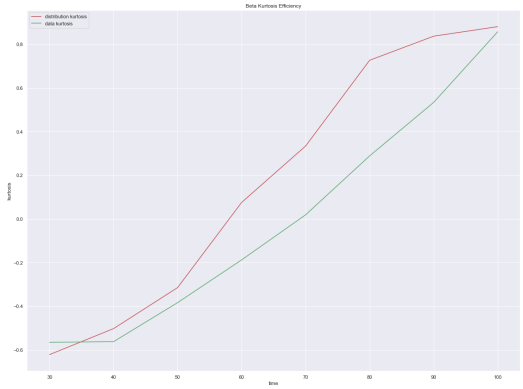


Figure 3.25: Kurtosis

3.3 Case II: wind speed 10 km/h

- **Data separation and data cleansing:** Given the output matrix X, it has been split in the different matrices.

X0[1000]	t=0	X6[958]	t=60
X1[917]	t=10	X7[949]	t=70
X2[962]	t=20	X8[949]	t=80
X3[961]	t=30	X9[949]	t=90
X4[958]	t=40	X10[949]	t=100
X5[958]	t=50		

Table 3.2: The X matrix is divided into column vectors representing the number of threads for each time t.

- **Histogram representation:** Shown below the histograms for each time step with the correspondent empirical distribution.

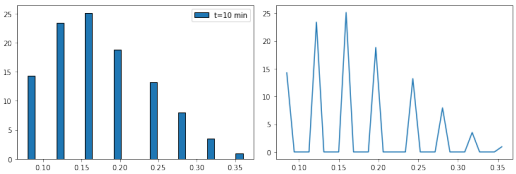


Figure 3.26: $X1[917],\ t = 10.$

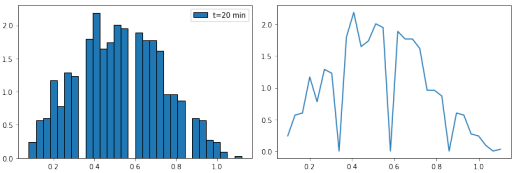


Figure 3.27: $X2[962],\ t = 20.$

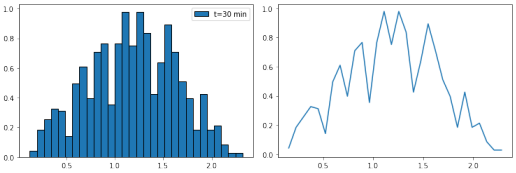


Figure 3.28: $X3[961],\ t = 30.$

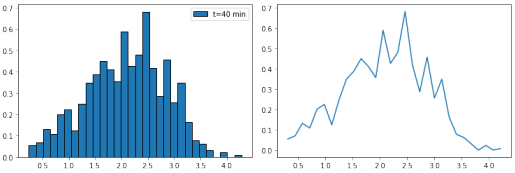


Figure 3.29: $X4[958],\ t = 40.$

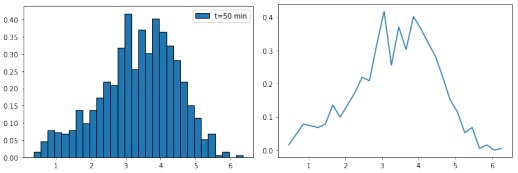


Figure 3.30: $X5[958],\ t = 50.$

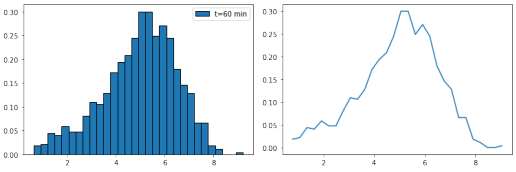


Figure 3.31: $X6[958],\ t = 60.$

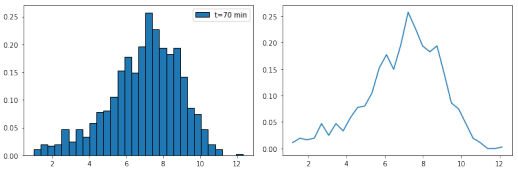


Figure 3.32: $X7[958],\ t = 70.$

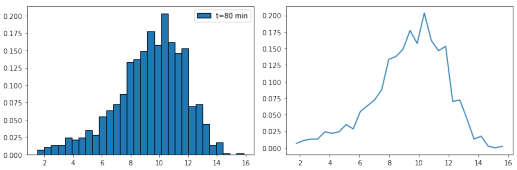


Figure 3.33: $X8[958],\ t = 80.$

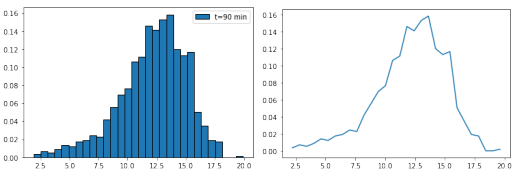


Figure 3.34: $X9[958],\ t = 90.$

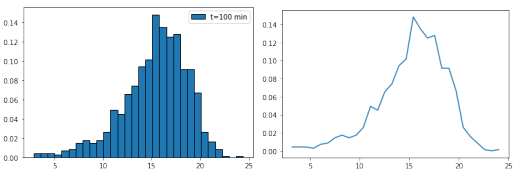


Figure 3.35: $X10[958],\ t = 100.$

The histograms and the empirical distributions obtained for each individual vector are plotted on the same graph to observe its trend over time:

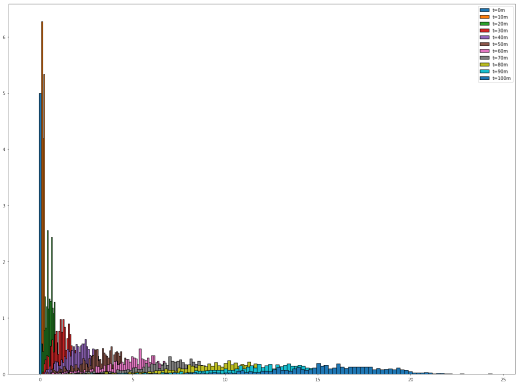


Figure 3.36: Histograms trend over the time

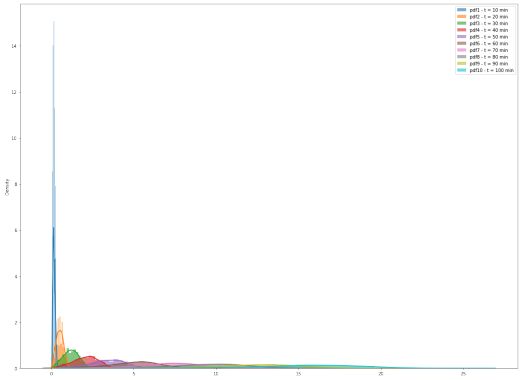


Figure 3.37: Distribution trend over the time

Considering the variable $X - A$, i.e. area for each tread minus A mean, a change in the trend and in the accumulation point can be observed:

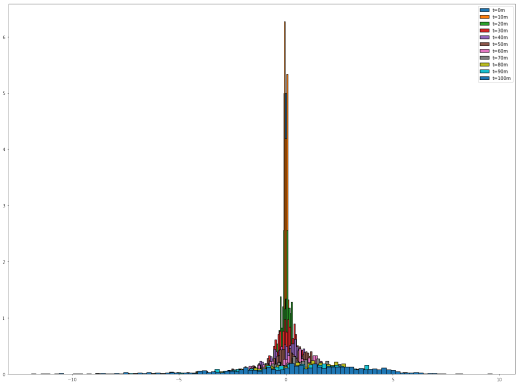


Figure 3.38: Histograms trend over the time

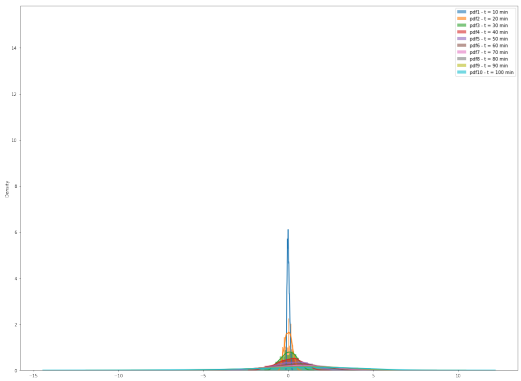


Figure 3.39: Distribution trend over the time

- **Study of trends:**

  Considering the outputs, we study the trends of mean, variance, correlation, maximum of the distribution and last point. Note that maximum of the distribution means the element belonging to the dataset to which the highest point of the curve corresponds.

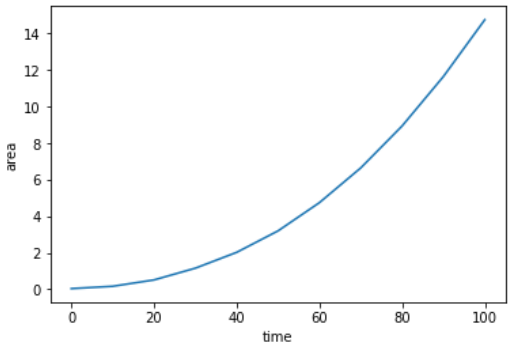


Figure 3.40: Mean Area trend

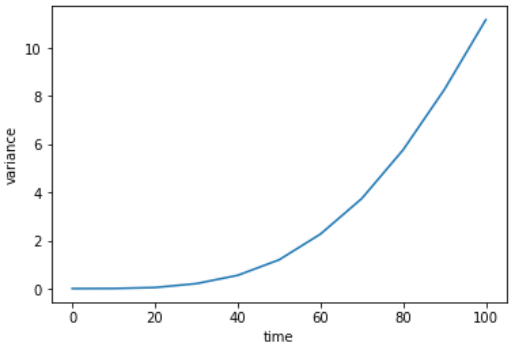


Figure 3.41: Variance trend

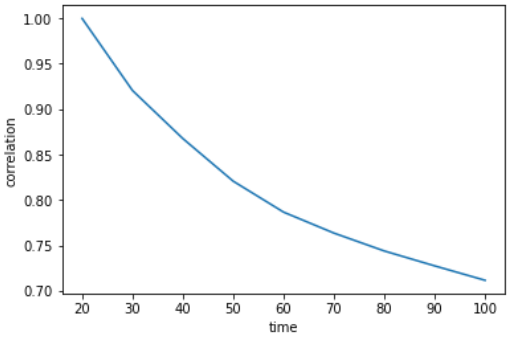


Figure 3.42: Correlation trend

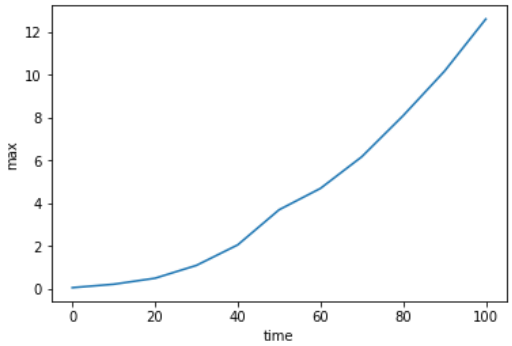


Figure 3.43: Trend of the maximum of the pdf

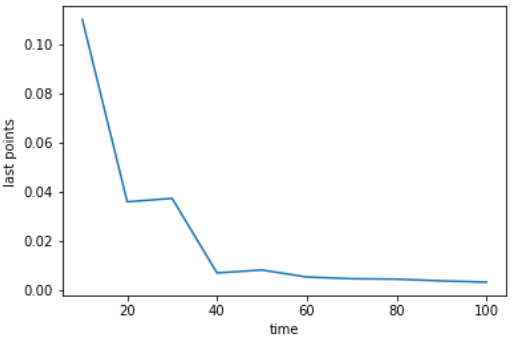


Figure 3.44: Last point trend

- **Data scaling:** To proceed with the analysis we now consider the scaled data, therefore we calculate

$$Y = \frac{X - A}{\sqrt{var(X)}} \tag{3.2}$$

as well as the X matrix, the Y matrix is divided into column vectors corresponding to the instants of observation.
We conducted the same analysis reported above also for the Y vectors but in order to focus the attention on the successive results, we do not show it.

- **Parameters calculation and data fitting:** Considering the variable Y, our goal is to find the parameters for it to be a beta distribution. For this purpose, we can use the *beta.fit* Python function (Appendix B) which calculate, in an empirically way, the parameters α and β.
In order to use *beta.fit* it is necessary a data scaling in the interval $[0, 1]$.
Using the obtained values of the needed parameters, a fit of the data is performed obtaining the following plots:

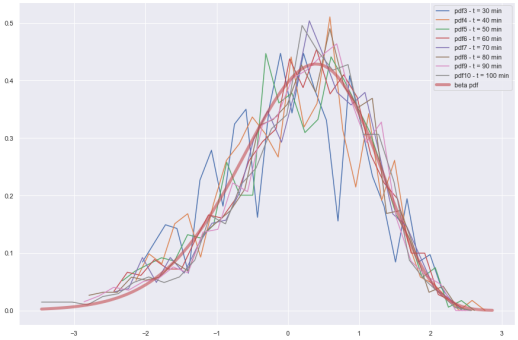


Figure 3.45: Linear beta fit

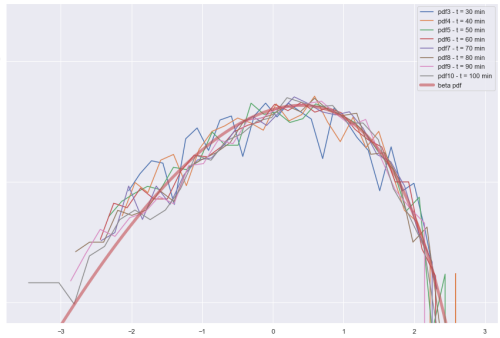


Figure 3.46: Logarithmic beta fit

- **Analysis of moments of various order:** Considering the four moments (mean, variance, skewness and kurtosis), they are calculated on the scaled Y vectors and their distribution. The results are shown in the following plots:

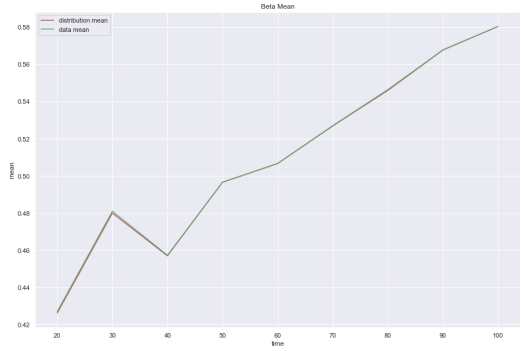


Figure 3.47: Mean

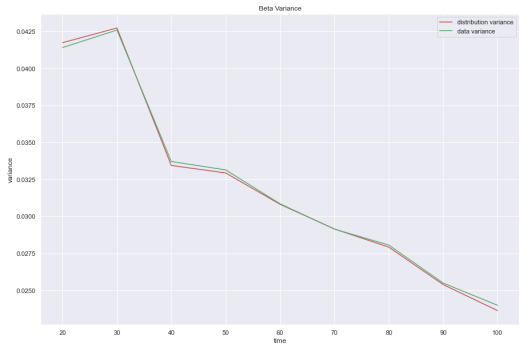


Figure 3.48: Variance

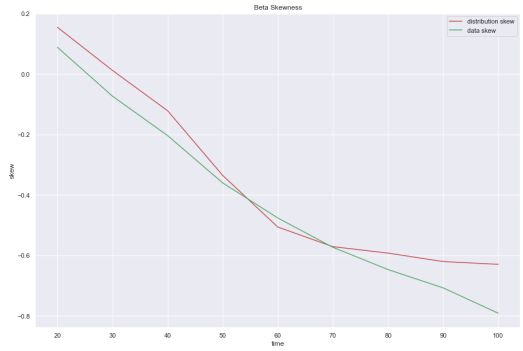


Figure 3.49: Skewness

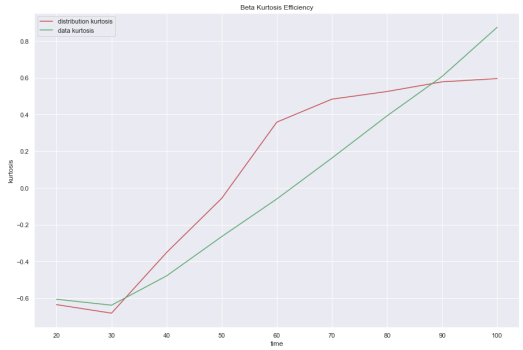


Figure 3.50: Kurtosis

3.4 Case III: wind speed 20 km/h

- **Data separation and data cleansing:** Given the output matrix X, it has been split in the different matrices.

X0[1000]	t=0	X6[970]	t=60
X1[931]	t=10	X7[970]	t=70
X2[969]	t=20	X8[970]	t=80
X3[970]	t=30	X9[970]	t=90
X4[970]	t=40	X10[970]	t=100
X5[970]	t=50		

Table 3.3: The X matrix is divided into column vectors representing the number of threads for each time t.

- **Histogram representation:** Shown below the histograms for each time step with the correspondent empirical distribution.

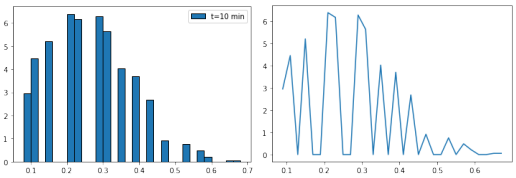


Figure 3.51: $X1[931],\ t = 10.$

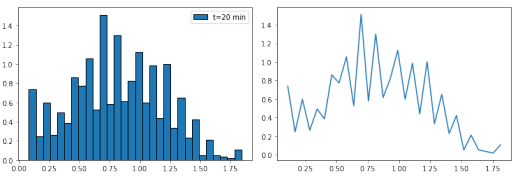


Figure 3.52: $X2[969],\ t = 20.$

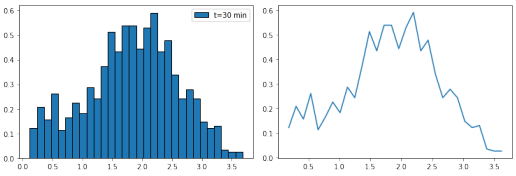


Figure 3.53: $X3[970],\ t = 30.$

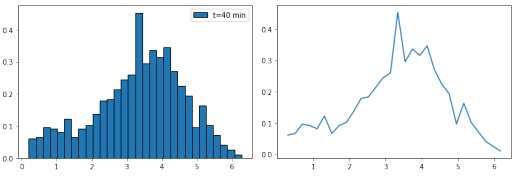


Figure 3.54: $X4[970],\ t = 40.$

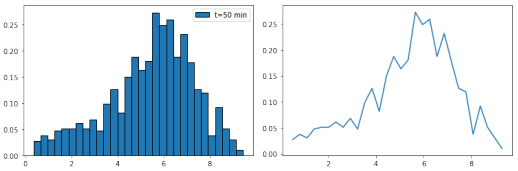


Figure 3.55: $X5[970],\ t = 50.$

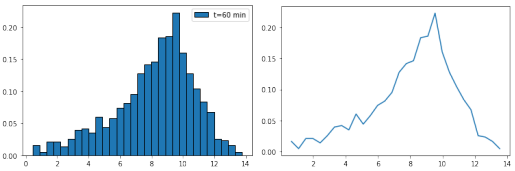


Figure 3.56: $X6[970],\ t = 60.$

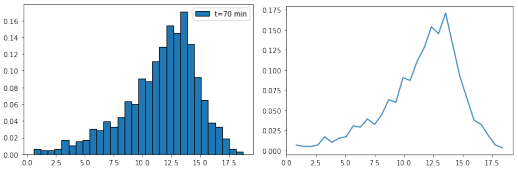


Figure 3.57: $X7[970],\ t = 70.$

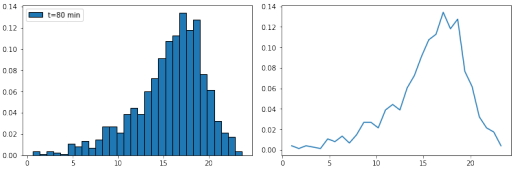


Figure 3.58: $X8[970],\ t = 80.$

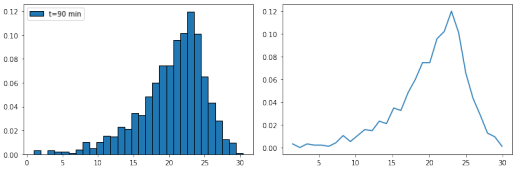


Figure 3.59: $X9[970],\ t = 90.$

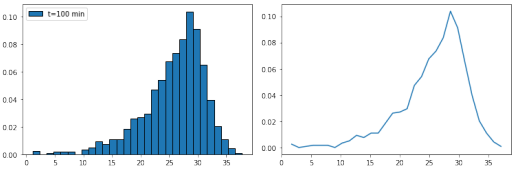


Figure 3.60: $X10[970],\ t = 100.$

The histograms and the empirical distributions obtained for each individual vector are plotted on the same graph to observe its trend over time:

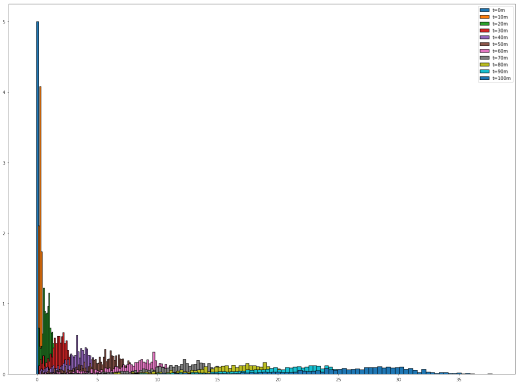


Figure 3.61: Histograms trend over the time

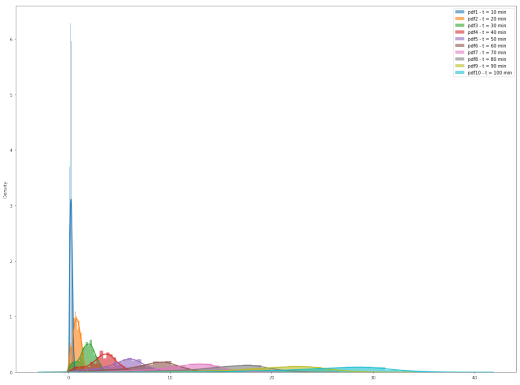


Figure 3.62: Distribution trend over the time

Considering the variable $X - A$, i.e. area for each tread minus A mean, a change in the trend and in the accumulation point can be observed:

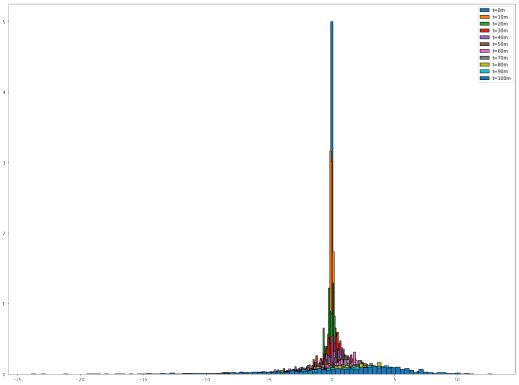


Figure 3.63: Histograms trend over the time

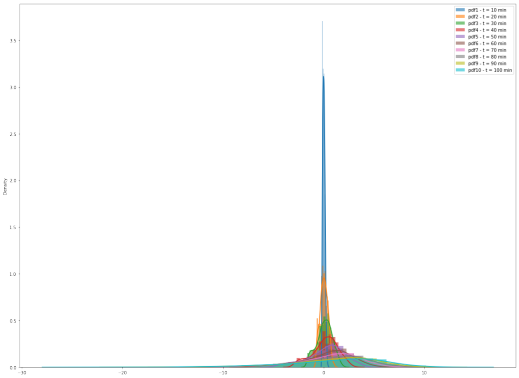


Figure 3.64: Distribution trend over the time

- **Study of trends:**

  Considering the outputs, we study the trends of mean, variance, correlation, maximum of the distribution and last point. Note that maximum of the distribution means the element belonging to the dataset to which the highest point of the curve corresponds.

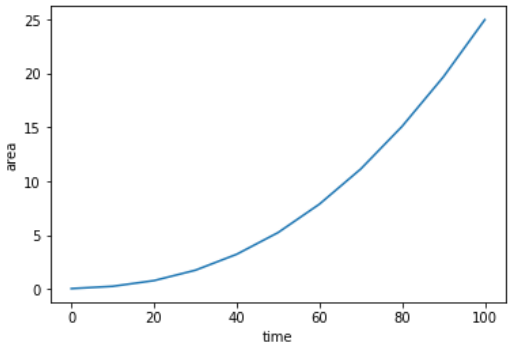


Figure 3.65: Mean Area trend

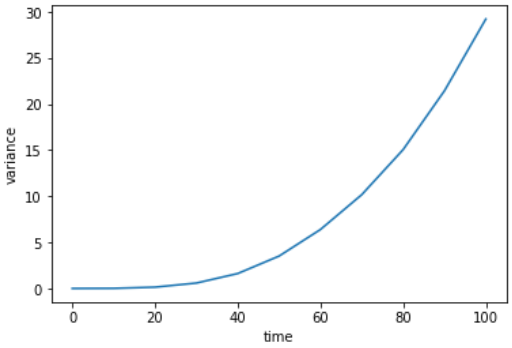


Figure 3.66: Variance trend

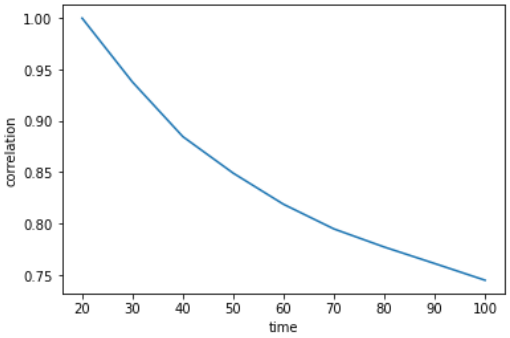


Figure 3.67: Correlation trend

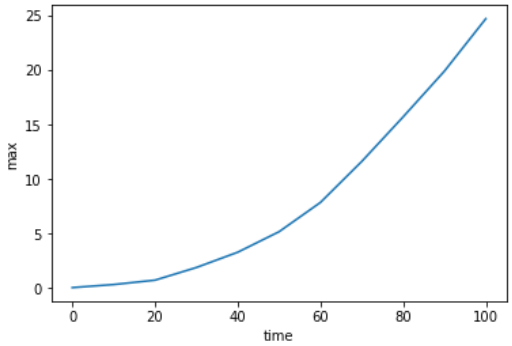


Figure 3.68: Trend of the maximum of the pdf

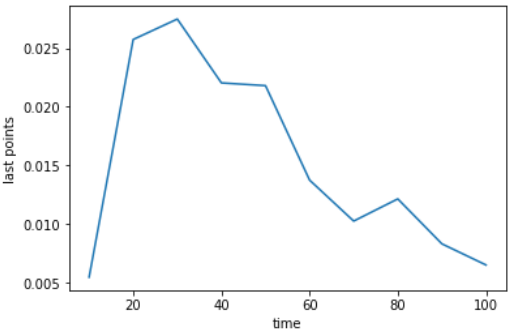


Figure 3.69: Last point trend

- **Data scaling:** To proceed with the analysis we now consider the scaled data, therefore we calculate

$$Y = \frac{X - A}{\sqrt{var(X)}} \tag{3.3}$$

as well as the X matrix, the Y matrix is divided into column vectors corresponding to the instants of observation.
We conducted the same analysis reported above also for the Y vectors but in order to focus the attention on the successive results, we do not show it.

- **Parameters calculation and data fitting:** Considering the variable Y, our goal is to find the parameters for it to be a beta distribution. For this purpose, we can use the *beta.fit* Python function (Appendix B) which calculate, in an empirically way, the parameters α and β.
In order to use *beta.fit* it is necessary a data scaling in the interval $[0, 1]$.
Using the obtained values of the needed parameters, a fit of the data is performed obtaining the following plots:

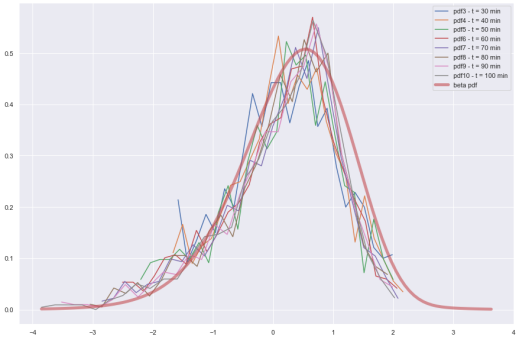


Figure 3.70: Linear beta fit

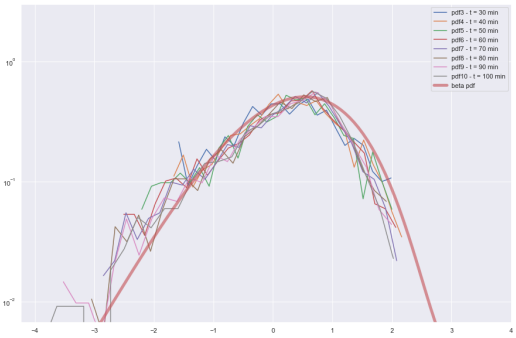


Figure 3.71: Logarithmic beta fit

- **Analysis of moments of various order:** Considering the four moments (mean, variance, skewness and kurtosis), they are calculated on the scaled Y vectors and their distribution. The results are shown in the following plots:

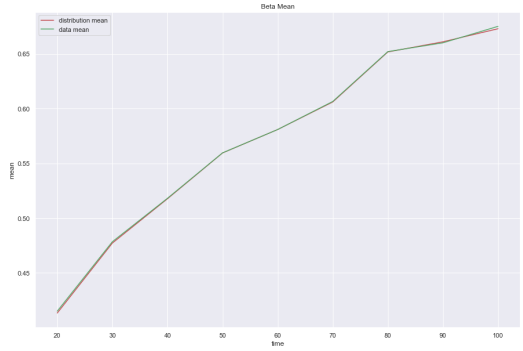


Figure 3.72: Mean

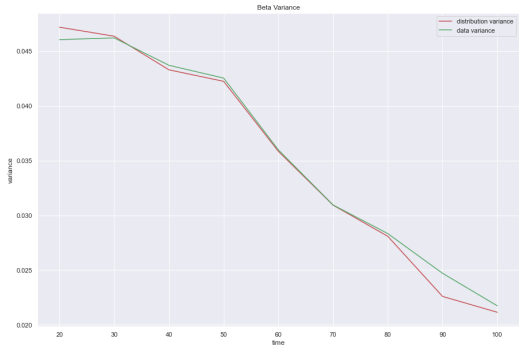


Figure 3.73: Variance

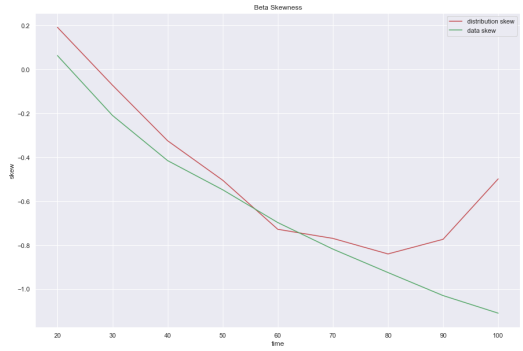


Figure 3.74: Skewness

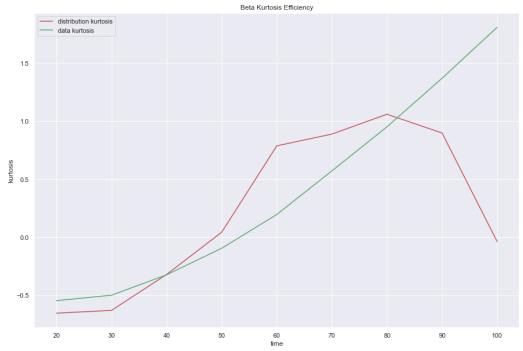


Figure 3.75: Kurtosis

3.5 Case IV: wind speed 30 km/h

- **Data separation and data cleansing:** Given the output matrix X, it has been split in the different matrices.

X0[1000]	t=0	X6[973]	t=60
X1[918]	t=10	X7[971]	t=70
X2[887]	t=20	X8[970]	t=80
X3[965]	t=30	X9[973]	t=90
X4[961]	t=40	X10[973]	t=100
X5[966]	t=50		

Table 3.4: The X matrix is divided into column vectors representing the number of threads for each time t.

- **Histogram representation:** Shown below the histograms for each time step with the correspondent empirical distribution.

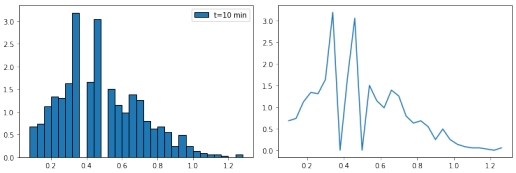


Figure 3.76: $X1[918],\ t = 10.$

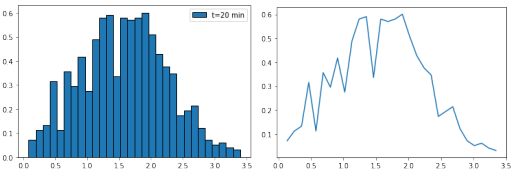


Figure 3.77: $X2[887],\ t = 20.$

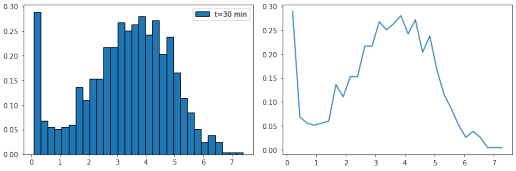


Figure 3.78: $X3[965],\ t = 30.$

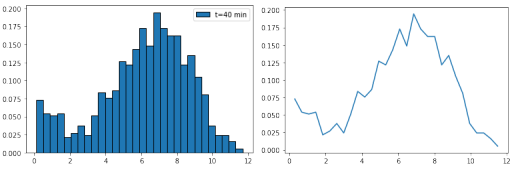


Figure 3.79: $X4[961],\ t = 40.$

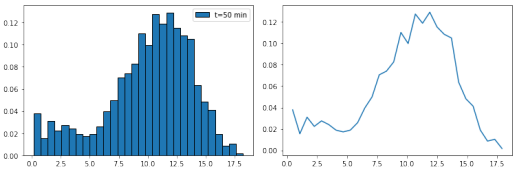


Figure 3.80: $X5[966],\ t = 50.$

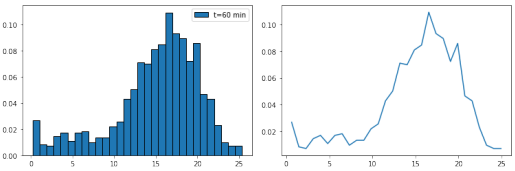


Figure 3.81: $X6[973],\ t = 60.$

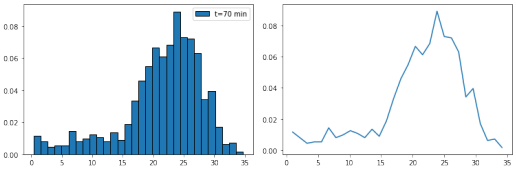


Figure 3.82: $X7[971],\ t = 70.$

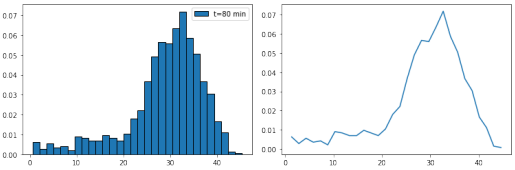


Figure 3.83: $X8[970],\ t = 80.$

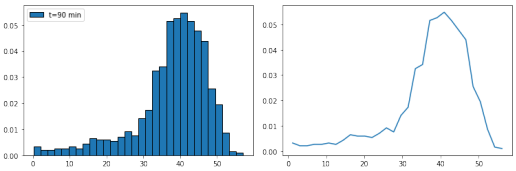


Figure 3.84: $X9[973],\ t = 90.$

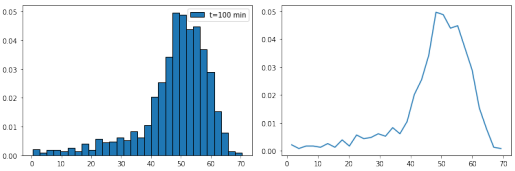


Figure 3.85: $X10[973],\ t = 100.$

The histograms and the empirical distributions obtained for each individual vector are plotted on the same graph to observe its trend over time:

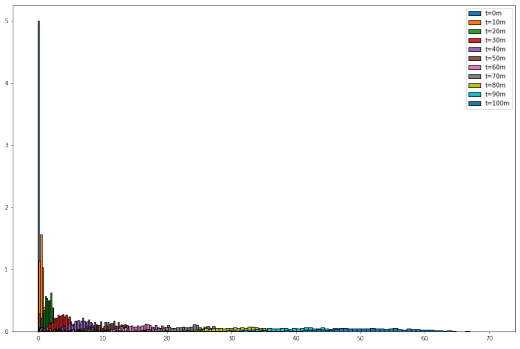


Figure 3.86: Histograms trend over the time

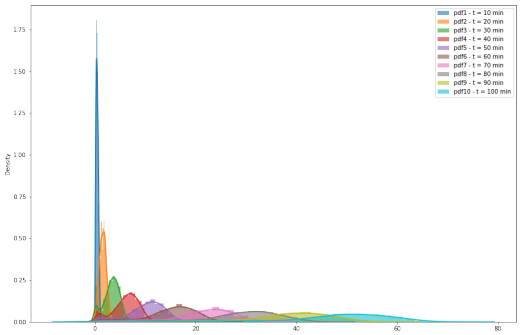


Figure 3.87: Distribution trend over the time

Considering the variable $X - A$, i.e. area for each tread minus A mean, a change in the trend and in the accumulation point can be observed:

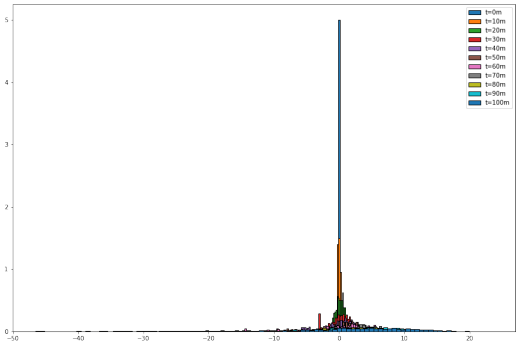


Figure 3.88: Histograms trend over the time

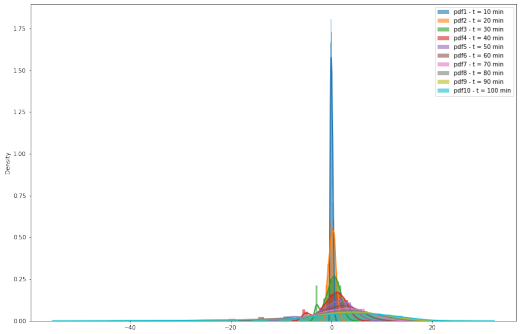


Figure 3.89: Distribution trend over the time

- **Study of trends:**

  Considering the outputs, we study the trends of mean, variance, correlation, maximum of the distribution and last point. Note that maximum of the distribution means the element belonging to the dataset to which the highest point of the curve corresponds.

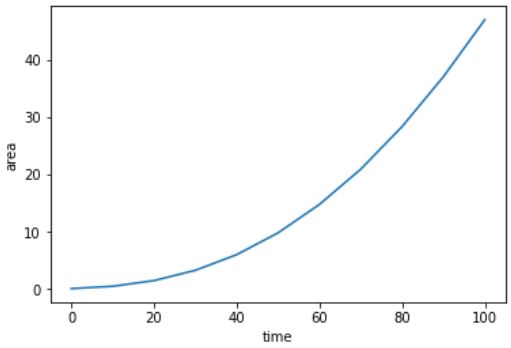


Figure 3.90: Mean Area trend

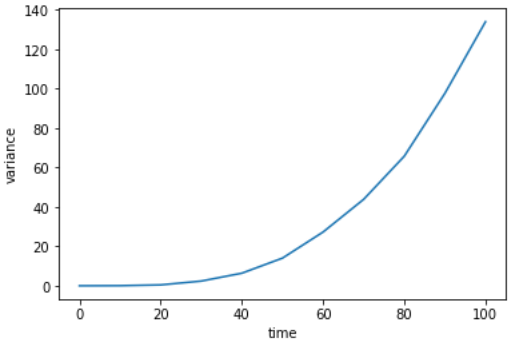


Figure 3.91: Variance trend

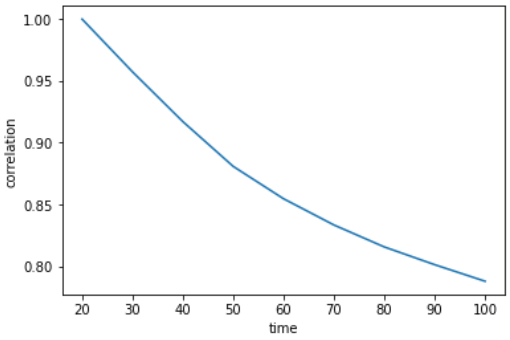


Figure 3.92: Correlation trend

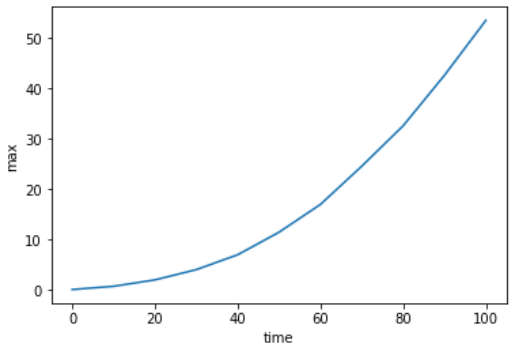


Figure 3.93: Trend of the maximum of the pdf

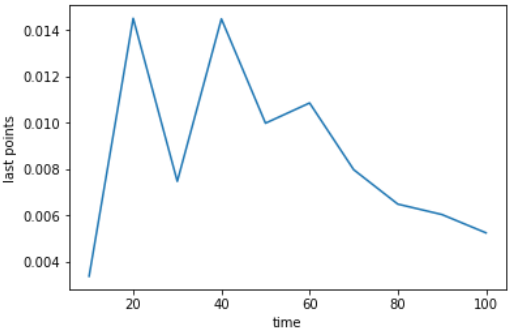


Figure 3.94: Last point trend

- **Data scaling:** To proceed with the analysis we now consider the scaled data, therefore we calculate

$$Y = \frac{X - A}{\sqrt{var(X)}} \tag{3.4}$$

as well as the X matrix, the Y matrix is divided into column vectors corresponding to the instants of observation.
We conducted the same analysis reported above also for the Y vectors but in order to focus the attention on the successive results, we do not show it.

- **Parameters calculation and data fitting:** Considering the variable Y, our goal is to find the parameters for it to be a beta distribution. For this purpose, we can use the *beta.fit* Python function (Appendix B) which calculate, in an empirically way, the parameters α and β.
In order to use *beta.fit* it is necessary a data scaling in the interval $[0, 1]$.
Using the obtained values of the needed parameters, a fit of the data is performed obtaining the following plots:

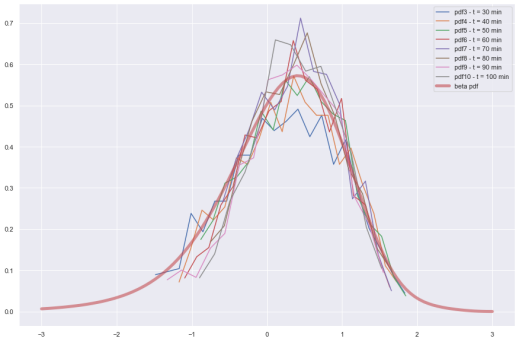


Figure 3.95: Linear beta fit

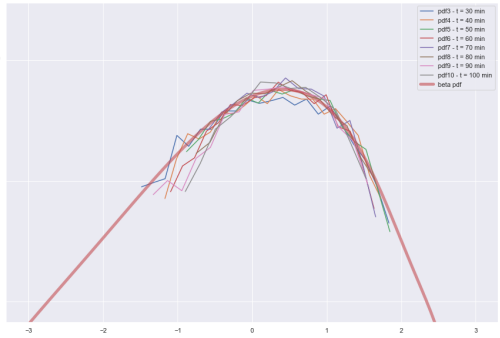


Figure 3.96: Logarithmic beta fit

- **Analysis of moments of various order:** Considering the four moments (mean, variance, skewness and kurtosis), they are calculated on the scaled Y vectors and their distribution. The results are shown in the following plots:

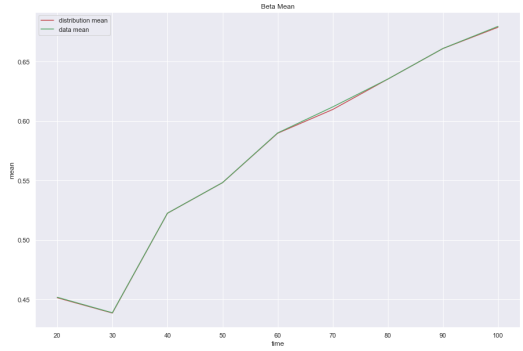


Figure 3.97: Mean

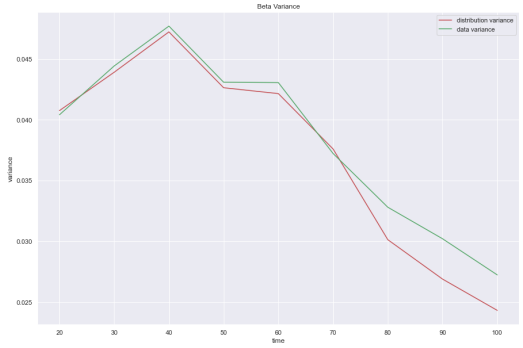


Figure 3.98: Variance

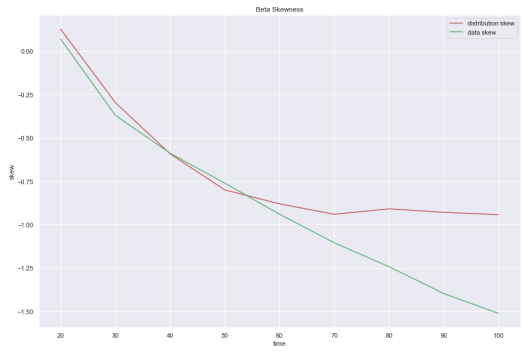


Figure 3.99: Skewness

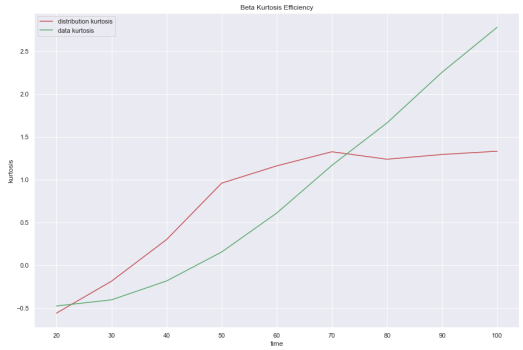


Figure 3.100: Kurtosis

Conclusions

The objective of the thesis was to identify how the burnt area is distributed over a limited observation interval. To this aim, a simplified fire propagation case was considered.

To achieve the main goal of the work, it was necessary to modify the PROPAGATOR algorithm. The change allowed us to have as output the burnt area values for each realization, for each instant of time. This modification, necessary for greater accuracy of the analysis, cost in terms of computational effort and since the study was made on 1000 realizations, it was necessary to use the server *Hypatia* made available by the BCAM-Basque Center for Applied Mathematics, since our computers supported the use of the Propagator algorithm for a maximum of 100 realizations.

As mentioned, the subject of the analysis are values representing areas, so it is evident that we are talking about a closed and limited domain. For an easier understanding: we are studying a wildfire, thus a phenomenon that has a limited duration in time and therefore the area subtended by the curve describing its propagation is also limited.
Hence the difficulty encountered was: finding a distribution with a limited domain that describes the propagation of this process. The most of the known statistical tests for parameter estimation are usable for unlimited domain distributions, hence our choice to do the analysis by direct comparisons.

What we did was, by looking at the propagation trend, notice that the beta distribution could be a good approximation of it. Then, to validate the hypothesis, moments of different order were calculated and compared.
As can be seen from the plots in the discussion of the analysis, for *mean* and *variance* the trends are almost similar, whereas for *skewness* and *kurtosis* it is not possible to say the same.
The reason for this difference is an error due to the continuous rescaling of the data. There is an accumulation of this type of error and a consequently

repercussion in the calculation of these two moments.
What can be concluded is that certainly for short times the beta function is suitable for such a comparison, but for long times the same cannot be said with certainty.

Given the phenomenon under consideration, it was realised that a simple mathematical model based on differential equations would not have been suitable for describing the problem, as it was quite complex. Therefore, a *modelling approach* to the phenomenon was necessary, resulting in the choice of a probabilistic cellular automata model, which proved to be more flexible, suitable and rapid. This represented, for us, a new method for modelling a physical phenomenon.

Developing this thesis was not only carrying out a scientific work but, in order to achieve these results, it was necessary to expand our wealth of knowledge. We dealt with writing new routines in Python and statistical in-depth analysis, topics which were not furthered during the University lessons.

Moreover, the scientific growth come up beside the personal development. Working in an international environment let people deal with new cultures and realities. To grow up and move from the University into the wider research areas, entails to plunge into a world full of obstacles and mistakes.

Appendix A

Codes

In this appendix we report all the lines of code we wrote to carry out the analysis we discuss in the chapter 3.

The codes are written for a general case in fact we write i in place of the particular matrix considered. All the analysis reported here has been done in the same way for the different four cases: wind equal to 0 km/h, 10 km/h 20 km/h, 30 km/h, considering the corresponding data.

Data separation e cleansing

```
import numpy as np
import pandas as pd

dati = 'dati_1000.csv'   # data set 1000 threads
CSVData = open(dati)

X = np.loadtxt(CSVData, delimiter=",")  # we are putting
                                        # all the data in a matrix

# data separation

X1=X[1002:2002,0]
X2=X[2003:3003,0]
X3=X[3004:4004,0]
X4=X[4005:5005,0]
X5=X[5006:6006,0]
X6=X[6007:7007,0]
```

```
X7=X[7008:8008,0]
X8=X[8009:9009,0]
X9=X[9010:10010,0]
X10=X[10011:11011,0]

# data cleansing

def no_repertir(X_curr, X_prev):
    dif_x = X_curr-X_prev
    X_curr = X_curr[dif_x > np.exp(-9)]

    return X_curr

X1 = no_repertir(X1, X0)
X2 = no_repertir(X2, X1)
X3 = no_repertir(X3, X2)
X4 = no_repertir(X4, X3)
X5 = no_repertir(X5, X4)
X6 = no_repertir(X6, X5)
X7 = no_repertir(X7, X6)
X8 = no_repertir(X8, X7)
X9 = no_repertir(X9, X8)
X10 = no_repertir(X10, X9)
```

A.1 Histogram representation

```
import matplotlib.pyplot as plt
import seaborn as sns

Ni,Bi,_=plt.hist(Xi,bins=30,density=True,edgecolor='black',
                 label='t=10 min')
plt.legend()
plt.show()

#pdf

sns.distplot(Xi,bins=30,color="blue",hist_kws=dict(density=True,
             edgecolor="black",linewidth=1,label='t=10 min'))
plt.legend()
plt.show()

```

```
sns.distplot(Xi,bins=30,color="green",hist_kws=dict(density=True,
            edgecolor="black",linewidth=1,label='t=10 min'))
plt.legend()
plt.semilogy()
plt.show()

#empirical distribution

Ci = 0.5*(B1[1:]+B1[:-1])
plt.plot(C1,N1) ## using bin_centers rather than edges
plt.show()
```

```
from matplotlib import rcParams

rcParams['figure.figsize'] = 20, 15
plt.hist(X0,bins = 5,density=True,edgecolor='black',label='t=0m')
plt.hist(X1,bins = 5,density=True,edgecolor='black',label='t=10m')
plt.hist(X2,bins = 20,density=True,edgecolor='black',label='t=20m')
plt.hist(X3,bins = 30,density=True,edgecolor='black',label='t=30m')
plt.hist(X4,bins = 50,density=True,edgecolor='black',label='t=40m')
plt.hist(X5,bins = 80,density=True,edgecolor='black',label='t=50m')
plt.hist(X6,bins = 80,density=True,edgecolor='black',label='t=60m')
plt.hist(X7,bins = 80,density=True,edgecolor='black',label='t=70m')
plt.hist(X8,bins = 80,density=True,edgecolor='black',label='t=80m')
plt.hist(X9,bins = 90,density=True,edgecolor='black',label='t=90m')
plt.hist(X10,bins = 100,density=True,edgecolor='black',label='t=100m')
plt.legend()
plt.show()
```

```
from matplotlib import rcParams

rcParams['figure.figsize'] = 20, 15
kwargs = dict(hist_kws={'alpha':.6}, kde_kws={'linewidth':2})

sns.distplot(X1,  label='pdf1 - t = 10 min', **kwargs)
sns.distplot(X2,  label='pdf2 - t = 20 min', **kwargs)
sns.distplot(X3,  label='pdf3 - t = 30 min', **kwargs)
sns.distplot(X4,  label='pdf4 - t = 40 min', **kwargs)
sns.distplot(X5,  label='pdf5 - t = 50 min', **kwargs)
sns.distplot(X6,  label='pdf6 - t = 60 min', **kwargs)
sns.distplot(X7,  label='pdf7 - t = 70 min', **kwargs)
sns.distplot(X8,  label='pdf8 - t = 80 min', **kwargs)
sns.distplot(X9,  label='pdf9 - t = 90 min', **kwargs)
```

```
sns.distplot(X10, label='pdf10 - t = 100 min', **kwargs)
#plt.plot(max, label='max')
plt.legend()
plt.show()
```

A.2 Study of trends

Mean Area

```
area_mean = 'areamean_wind20_1000.csv'
CSVData = open(area_mean)

A = np.loadtxt(CSVData, delimiter=",")
y=A[:,0]
x=A[:,1]

fig = plt.figure()
plt.plot(x,y)
fig.suptitle('mean area')
plt.xlabel('time')
plt.ylabel('area')
plt.show()
```

Variance

```
from statistics import variance

# the variance has been calculated with
# python function, following the formula:
# Var(X)=E[(X-mu)^2]

V0=variance(X0)
V1=variance(X1)
V2=variance(X2)
V3=variance(X3)
V4=variance(X4)
V5=variance(X5)
V6=variance(X6)
V7=variance(X7)
V8=variance(X8)
```

```
V9=variance(X9)
V10=variance(X10)

V=[V0, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10]
V = np.array(V)
print(V)

t=A[:,1]

fig = plt.figure()
plt.plot(t,V)
fig.suptitle('variance trend')
plt.xlabel('time')
plt.ylabel('variance')
plt.show()
```

Correlation

```
from scipy.stats import pearsonr

# function which calculate the Pearson correlation coefficient:
# output: r correlation coefficient;
#         two-sided p-value

# s fixed to the second time step

#corr21=np.array(pearsonr(X[2003:3003,0],X[1:1001,0]))
corr22=np.array(pearsonr(X[2003:3003,0],X[2003:3003,0]))
corr23=np.array(pearsonr(X[2003:3003,0],X[3004:4004,0]))
corr24=np.array(pearsonr(X[2003:3003,0],X[4005:5005,0]))
corr25=np.array(pearsonr(X[2003:3003,0],X[5006:6006,0]))
corr26=np.array(pearsonr(X[2003:3003,0],X[6007:7007,0]))
corr27=np.array(pearsonr(X[2003:3003,0],X[7008:8008,0]))
corr28=np.array(pearsonr(X[2003:3003,0],X[8009:9009,0]))
corr29=np.array(pearsonr(X[2003:3003,0],X[9010:10010,0]))
corr210=np.array(pearsonr(X[2003:3003,0],X[10011:11011,0]))

corr2=np.array([corr22[0],corr23[0],corr24[0],corr25[0],
        corr26[0],corr27[0],corr28[0],corr29[0],corr210[0]])
print(corr2)
t=A[:,1]
```

```
fig = plt.figure()
plt.plot(t[2:11],corr2)
fig.suptitle('correlation trend')
plt.xlabel('time')
plt.ylabel('correlation')
plt.show()
```

Maximum of the pdf

```
from scipy.stats import norm

norm_dist_Xi = norm(Xi.mean(), Xi.std())
xi = np.linspace(np.min(Xi, np.max(Xi), 1000)
pdf_Xi = [norm_dist_Xi.pdf(x) for x in xi]
pdf_Xi = np.array(pdf_Xi)
dxi=pdf_Xi[999]
maxi=np.max(pdf_Xi)

while True:
    for x in Xi:
      pdf_maxi = np.array([norm_dist_Xi.pdf(x)])
      maxxi=x
    if not abs(maxi-pdf_maxi)<np.exp(-7):
        break

maxx=np.array([X0[1],maxx1,maxx2,maxx3,maxx4,maxx5,
                maxx6,maxx7,maxx8,maxx9,maxx10])
fig = plt.figure()
plt.plot(t,maxx)
fig.suptitle('max point trend')
plt.xlabel('time')
plt.ylabel('max')
plt.show()
```

Last point

```
dx=np.array([dx0,dx1,dx2,dx3,dx4,dx5,dx6,dx7,dx8,dx9,dx10])
print(dx)
#plt.rcdefaults()

dx=np.array([dx1,dx2,dx3,dx4,dx5,dx6,dx7,dx8,dx9,dx10])
```

```
fig = plt.figure()
plt.plot(t[1:11],dx)
fig.suptitle('last point trend')
plt.xlabel('time')
plt.ylabel('last points')
plt.show()
```

A.3 Data scaling

```
# now we consider the variable Y=A-Amean/sqrt(var(A))

tot=np.concatenate((Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,Y10),axis=None)

# data scaling

import numpy as np

def NormalizeData(data):
    return (data-np.min(data)) / (np.max(data)-np.min(data))

scaled_tot = NormalizeData(tot)
```

A.4 Parameters calculation and data fitting

Beta distribution

```

from sklearn.datasets import load_diabetes
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import pandas as pd
from distfit import distfit
from scipy.stats import beta

#loc is short for "location parameter",
# and scale is any scale parameter
# location parameters would include the mean in the
# normal distribution and the median in the Cauchy distribution
# scale parameters are like the standard deviation in the normal
# distribution, or either parameter of the gamma distribution
```

```
15
a, b, loc, scale = beta.fit(scaled_tot)
17 print(a, b, loc, scale)
ax = plt.subplot(111)
19 ax.plot(np.linspace(0, 1, 100), beta.pdf(np.linspace(0, 1, 100),
        a, b, loc, scale))
21 plt.show()

23 l=[min(CY1), min(CY2), min(CY3), min(CY4), min(CY5), min(CY6),
    min(CY7), min(CY8), min(CY9), min(CY10)]
25 g=[max(CY1), max(CY2), max(CY3), max(CY4), max(CY5), max(CY6),
    max(CY7), max(CY8), max(CY9), max(CY10)]
27 mm=min(l)
MM=max(g)
29

import numpy as np
31 import matplotlib.pyplot as plt
from scipy.stats import beta
33


35 fig, ax = plt.subplots(1, 1)
x=np.linspace(0,1,100)
37 y= mm + x*(MM-mm)
z=(y-mm)/(MM-mm)
39 ax.plot(y, beta.pdf(z, a, b, loc, scale)*[1/(MM-mm)],
        'r-', lw=5, alpha=0.6, label='beta␣pdf')
```

```
1

plt.plot(CY2,NY2_new,label='pdf2␣-␣t␣=␣20␣min')
3 plt.plot(CY3,NY3_new,label='pdf3␣-␣t␣=␣30␣min')
plt.plot(CY4,NY4_new,label='pdf4␣-␣t␣=␣40␣min')
5 plt.plot(CY5,NY5_new,label='pdf5␣-␣t␣=␣50␣min')
plt.plot(CY6,NY6_new,label='pdf6␣-␣t␣=␣60␣min')
7 plt.plot(CY7,NY7_new,label='pdf7␣-␣t␣=␣70␣min')
plt.plot(CY8,NY8_new,label='pdf8␣-␣t␣=␣80␣min')
9 plt.plot(CY9,NY9_new,label='pdf9␣-␣t␣=␣90␣min')
plt.plot(CY10,NY10_new,label='pdf10␣-␣t␣=␣100␣min')
11 plt.plot(y, beta.pdf(z, a, b, loc, scale)*[1/(MM-mm)], 'r-',
        lw=5, alpha=0.6, label='beta␣pdf')
13

plt.title('Distribuzione␣di␣(A-Amedia)/sqrt[var(A)]')
15 plt.legend()
plt.show()
```

```
17
   # logarithmic
19
   plt.plot(CY2,NY2_new,label='pdf2 - t = 20 min')
21 plt.plot(CY3,NY3_new,label='pdf3 - t = 30 min')
   plt.plot(CY4,NY4_new,label='pdf4 - t = 40 min')
23 plt.plot(CY5,NY5_new,label='pdf5 - t = 50 min')
   plt.plot(CY6,NY6_new,label='pdf6 - t = 60 min')
25 plt.plot(CY7,NY7_new,label='pdf7 - t = 70 min')
   plt.plot(CY8,NY8_new,label='pdf8 - t = 80 min')
27 plt.plot(CY9,NY9_new,label='pdf9 - t = 90 min')
   plt.plot(CY10,NY10_new,label='pdf10 - t = 100 min')
29 plt.plot(y, beta.pdf(z, a, b, loc, scale)*[1/(MM-mm)], 'r-',
       lw=5, alpha=0.6, label='beta pdf')
31

33 plt.semilogy()
   plt.title('Distribuzione di (A-Amedia)/sqrt[var(A)]')
35 plt.ylim(np.exp(-5),3)
   plt.legend()
37 plt.show()
```

The different beta distributions, which are plot in figures 3.20, 3.45, 3.70, 3.95, in order to have a better approximation, are obtained as a sum of beta distributions.

A.5 Analysis of moments of various order

```
   scaled_totY2 = NormalizeData(Y2)
2  scaled_totY3 = NormalizeData(Y3)
   scaled_totY4 = NormalizeData(Y4)
4  scaled_totY5 = NormalizeData(Y5)
   scaled_totY6 = NormalizeData(Y6)
6  scaled_totY7 = NormalizeData(Y7)
   scaled_totY8 = NormalizeData(Y8)
8  scaled_totY9 = NormalizeData(Y9)
   scaled_totY10 = NormalizeData(Y10)
10
   a_y2, b_y2, loc_y2, scale_y2 = beta.fit(scaled_totY2)
12 a_y3, b_y3, loc_y3, scale_y3 = beta.fit(scaled_totY3)
   a_y4, b_y4, loc_y4, scale_y4 = beta.fit(scaled_totY4)
14 a_y5, b_y5, loc_y5, scale_y5 = beta.fit(scaled_totY5)
```

```
a_y6, b_y6, loc_y6, scale_y6 = beta.fit(scaled_totY6)
a_y7, b_y7, loc_y7, scale_y7 = beta.fit(scaled_totY7)
a_y8, b_y8, loc_y8, scale_y8 = beta.fit(scaled_totY8)
a_y9, b_y9, loc_y9, scale_y9 = beta.fit(scaled_totY9)
a_y10, b_y10, loc_y10, scale_y10 = beta.fit(scaled_totY10)

mY2,vY2,sY2,kY2=beta.stats(a_y2,b_y2,loc_y2,scale_y2,moments='mvsk')
mY3,vY3,sY3,kY3=beta.stats(a_y3,b_y3,loc_y3,scale_y3,moments='mvsk')
mY4,vY4,sY4,kY4=beta.stats(a_y4,b_y4,loc_y4,scale_y4,moments='mvsk')
mY5,vY5,sY5,kY5=beta.stats(a_y5,b_y5,loc_y5,scale_y5,moments='mvsk')
mY6,vY6,sY6,kY6=beta.stats(a_y6,b_y6,loc_y6,scale_y6,moments='mvsk')
mY7,vY7,sY7,kY7=beta.stats(a_y7,b_y7,loc_y7,cale_y7,moments='mvsk')
mY8,vY8,sY8,kY8=beta.stats(a_y8,b_y8,loc_y8,scale_y8,moments='mvsk')
mY9,vY9,sY9,kY9=beta.stats(a_y9,b_y9,loc_y9,scale_y9,moments='mvsk')
mY10,vY10,sY10,kY10=beta.stats(a_y10,b_y10,loc_y10,
                    scale_y10,moments='mvsk')
```

Mean

```
MY2=np.mean(scaled_totS2)
MY3=np.mean(scaled_totS3)
MY4=np.mean(scaled_totS4)
MY5=np.mean(scaled_totS5)
MY6=np.mean(scaled_totS6)
MY7=np.mean(scaled_totS7)
MY8=np.mean(scaled_totS8)
MY9=np.mean(scaled_totS9)
MY10=np.mean(scaled_totS10)

mY=[mY2,mY3,mY4,mY5,mY6,mY7,mY8,mY9,mY10]
mY=np.array(mY)
print(mY)

MY=[MY2, MY3, MY4, MY5, MY6, MY7, MY8, MY9, MY10]
MY = np.array(MY)
print(MY)

t=A[:,1]

plt.plot(t[2:],mY, 'r-', label='distribution␣mean')
plt.plot(t[2:],MY, 'g-', label='data␣mean')
```

```
plt.xlabel('time')
plt.ylabel('mean')
plt.title('Beta Mean')
plt.legend()
plt.show()
```

Variance

```
from statistics import variance

VY2=np.var(scaled_totY2)
VY3=np.var(scaled_totY3)
VY4=np.var(scaled_totY4)
VY5=np.var(scaled_totY5)
VY6=np.var(scaled_totY6)
VY7=np.var(scaled_totY7)
VY8=np.var(scaled_totY8)
VY9=np.var(scaled_totY9)
VY10=np.var(scaled_totY10)

vY=[vY2,vY3,vY4,vY5,vY6,vY7,vY8,vY9,vY10]
vY=np.array(vY)
print(vY)

VY=[VY2, VY3, VY4, VY5, VY6, VY7, VY8, VY9, VY10]
VY = np.array(VY)
print(VY)

t=A[:,1]

plt.plot(t[2:],vY, 'r-', label='distribution variance')
plt.plot(t[2:],VY, 'g-', label='data variance')
plt.xlabel('time')
plt.ylabel('variance')
plt.title('Beta Variance')
plt.legend()
plt.show()
```

Skewness

```
from scipy.stats import skew

SY2=skew(scaled_totY2)
SY3=skew(scaled_totY3)
SY4=skew(scaled_totY4)
SY5=skew(scaled_totY5)
SY6=skew(scaled_totY6)
SY7=skew(scaled_totY7)
SY8=skew(scaled_totY8)
SY9=skew(scaled_totY9)
SY10=skew(scaled_totY10)

SY=[SY2, SY3, SY4, SY5, SY6, SY7, SY8, SY9, SY10]
SY = np.array(SY)
print(SY)

sY=[sY2,sY3,sY4,sY5,sY6,sY7,sY8,sY9,sY10]
sY=np.array(sY)
print(sY)

t=A[:,1]

plt.plot(t[2:],sY, 'r-', label='distribution skew')
plt.plot(t[2:],SY, 'g-', label='data skew')
plt.xlabel('time')
plt.ylabel('skew')
plt.title('Beta Skewness')
plt.legend()
plt.show()
```

Kurtosis

```
from scipy.stats import kurtosis

KY2=kurtosis(scaled_totY2)
KY3=kurtosis(scaled_totY3)
KY4=kurtosis(scaled_totY4)
KY5=kurtosis(scaled_totY5)
KY6=kurtosis(scaled_totY6)
KY7=kurtosis(scaled_totY7)
KY8=kurtosis(scaled_totY8)
```

```
KY9=kurtosis(scaled_totY9)
KY10=kurtosis(scaled_totY10)

KY=[KY2, KY3, KY4, KY5, KY6, KY7, KY8, KY9, KY10]
KY = np.array(KY)
print(KS)

kY=[kY2,kY3,kY4,kY5,kY6,kY7,kY8,kY9,kY10]
kY=np.array(kY)
print(kY)

t=A[:,1]

plt.plot(t[2:],kY, 'r-', label='distribution kurtosis')
plt.plot(t[2:],KY, 'g-', label='data kurtosis')
plt.xlabel('time')
plt.ylabel('kurtosis')
plt.title('Beta Kurtosis Efficiency')
plt.legend()
plt.show()
```

Appendix B

Software

In this Appendix, you can find some of the functions from Python library used for the analysis. This function are used in the codes reported in Appendix A.

- *plt.hist*: uses numpy.histogram to bin the data in x and count the number of values in each bin, then draws the distribution either as a BarContainer or Polygon.

  Parameters:

  - $x : (n,)$ array or sequence of $(n,)$ arrays
    Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length.
  - bins: int or sequence
    If bins is an integer, it defines the number of equal-width bins in the range. If bins is a sequence, it defines the bin edges, including the left edge of the first bin and the right edge of the last bin; in this case, bins may be unequally spaced. All but the last (righthand-most) bin is half-open.
  - range: tuple or None, default: None
    The lower and upper range of the bins. Lower and upper outliers are ignored. If not provided, range is $(x.min(), x.max())$. Range has no effect if bins is a sequence. If bins is a sequence or range is specified, autoscaling is based on the specified bin range instead of the range of x.
  - density: bool, default: False
    If True, draw and return a probability density: each bin will display the bin's raw count divided by the total number of counts and the bin width $(density = counts/(sum(counts) * np.diff(bins)))$,

so that the area under the histogram integrates to 1 ($np.sum(density * np.diff(bins)) == 1$). If stacked is also True, the sum of the histograms is normalized to 1.

- color: color or array-like of colors or None, default: None
  Color or sequence of colors, one per dataset. Default (None) uses the standard line color sequence.
- label: str or None, default: None
  String, or sequence of strings to match multiple datasets. Bar charts yield multiple patches per dataset, but only the first gets the label, so that legend will work as expected.

Returns:

- n: array or list of arrays
  The values of the histogram bins. See density and weights for a description of the possible semantics. If input x is an array, then this is an array of length nbins. If input is a sequence of arrays [data1, data2, ...], then this is a list of arrays with the values of the histograms for each of the arrays in the same order. The dtype of the array n (or of its element arrays) will always be float even if no weighting or normalization is used.
- bins: array
  The edges of the bins. Length nbins + 1 (nbins left edges and right edge of last bin). Always a single array even when multiple data sets are passed in.
- patches:*BarContainer* or list of a single *Polygon* or list such objects
  Container of individual artists used to create the histogram or list of such containers if there are multiple input datasets.

- *sns.distplot*: a Distplot or distribution plot, depicts the variation in the data distribution. Seaborn Distplot represents the overall distribution of continuous data variables. The Seaborn module along with the Matplotlib module is used to depict the distplot with different variations in it. The Distplot depicts the data by a histogram and a line in combination to it.

Parameters:

- data: pandas.DataFrame, numpy.ndarray, mapping, or sequence
  Input data structure. Either a long-form collection of vectors that

can be assigned to named variables or a wide-form dataset that will be internally reshaped.

- label: str or None, default: None
  String, or sequence of strings to match multiple datasets. Bar charts yield multiple patches per dataset, but only the first gets the label, so that legend will work as expected.
- kwargs
  Other keyword arguments are documented with the relevant axes-level function.

Returns:
FaceGrid: an object managing one or more subplots that correspond to conditional data subsets with convenient methods for batch-setting of axes attributes.

- *nunpy.mean:* compute the arithmetic mean along the specified axis. Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis.

  Parameters:

  - a: array_like
    Array containing numbers whose mean is desired. If a is not an array, a conversion is attempted.
  - axis: None or int or tuple of ints, optional
    Axis or axes along which the means are computed. The default is to compute the mean of the flattened array.

  Returns:
  m: ndarray
  If out=None, returns a new array containing the mean values, otherwise a reference to the output array is returned.

- *variance*: returns the variance of the array elements, a measure of the spread of a distribution. The variance is computed for the flattened array by default, otherwise over the specified axis.

  Parameters:

  - a: array_like
    Array containing numbers whose mean is desired. If a is not an array, a conversion is attempted.

– axis: None or int or tuple of ints, optional
Axis or axes along which the means are computed. The default is to compute the mean of the flattened array.

Returns:
m: ndarray, see dtype parameter above
If out=None, returns a new array containing the mean values, otherwise a reference to the output array is returned.

- *correlation:* return Pearson product-moment correlation coefficients.The relationship between the correlation coefficient matrix, R, and the covariance matrix, C, is

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii}C_{jj}}}$$

Parameters:

– x: array_like
A 1-D or 2-D array containing multiple variables and observations. Each row of x represents a variable, and each column a single observation of all those variables.

– y: array_like
An additional set of variables and observations. y has the same shape as x.

Returns:
R: ndarray The correlation coefficient matrix of the variables.

- *beta fit:* given the input data, this function gives as outputs the parameters for the distribution to be a beta one. Therefore, it is possible to say that the distribution was calculated empirically.

Parameters: data: array_like

Returns:

– a, b: array_like
Shape parameters

– loc: array_like
Location parameter

– scale: array_like
Location parameter

- *beta.stats:* is a beta continuous random variable that is defined with a standard format and some shape parameters to complete its specification.

  Parameters: a, b, loc, scale

  Returns: mean, variance, skweness, kurtosis.

- *skewness:* for normally distributed data, the skewness should be about zero. For unimodal continuous distributions, a skewness value greater than zero means that there is more weight in the right tail of the distribution. The function skewtest can be used to determine if the skewness value is close enough to zero, statistically speaking.

  Parameters:

  - a: ndarray
    Input array
  - axis: int or None, default: 0
    If an int, the axis of the input along which to compute the statistic. The statistic of each axis-slice (e.g. row) of the input will appear in a corresponding element of the output. If None, the input will be raveled before computing the statistic.

  Returns:
  skewness: ndarray
  The skewness of values along an axis, returning NaN where all values are equal.

- *kurtosis:* compute the kurtosis (Fisher or Pearson) of a dataset. Kurtosis is the fourth central moment divided by the square of the variance. If Fisher's definition is used, then 3.0 is subtracted from the result to give 0.0 for a normal distribution. If bias is False then the kurtosis is calculated using k statistics to eliminate bias coming from biased moment estimators.

  Parameters:

  - a: ndarray
    Data for which the kurtosis is calculated.
  - axisint or None, default: 0
    If an int, the axis of the input along which to compute the statistic.

The statistic of each axis-slice (e.g. row) of the input will appear in a corresponding element of the output. If None, the input will be raveled before computing the statistic.

Returns:
kurtosis: array
The kurtosis of values along an axis, returning NaN where all values are equal.

Hypatia

Hypatia is the Cloud infrastructure that has been developed to support the computational needs of the ELIXIR-GR community, but also the broader community of life scientists in Greece and abroad. It currently hosts important ELIXIR-GR services and resources (e.g., the national COVID19 Data Portal of Greece), while it undertakes computational tasks in the context of various projects of ELIXIR-GR members. The infrastructure is named after Hypatia, a Greek philosopher, astronomer, and mathematician, who lived in Alexandria, Egypt.

Under the hood, Hypatia consists of a powerful computational cluster of heterogeneous physical machines. Currently, its cluster is comprised of: 32 basic nodes: (2 CPUs, 14 cores/CPU, 512GB DDR4 RAM), 2 hefty nodes: (2 CPUs, 24 cores/CPU, 1TB DDR4 RAM), 3 GPU nodes: (2 CPUs, 14 cores/CPU, 768GB DDR4 RAM, 2 GPUs), 8 I/O nodes: (2 CPUs, 14 cores/CPU, 512GB DDR4 RAM, 2×2TB SSD 6G), 9 infrastructure nodes:(2 CPUs, 14 cores/CPU, 192GB DDR4 RAM).

Hypatia's computational resources are allocated for predetermined time periods to particular user-created projects.
We had the opportunity to use this computational resource because it was provided by BCAM.

Appendix C

On Stochastic Processes

In this appendix there is a dissertation on stochastic processes carried out, in the first months of the project, as an approach and preparation for the subsequent work with Propagator.
The aim of these exercises was to get us used to working with such processes whose results are known so that we can then work with unknown processes. For further details, please refer to the texts [8], [16], [21], taken as sources.

C.1 Introduction to Stochastic Processes

A stochastic process is a mathematical object that describes dynamical systems whose time evolution is of a probabilistic nature, i.e they are phenomena that appear to vary in a random manner. In probability theory and related fields, a stochastic or random process is defined as a family of random variables.

Definition C.1 (Stochastic Process). Given an order set T, $(\Omega, \mathcal{F}, \mathbb{P})$ a probability space and $(E, \mathcal{G})$ a measurable space.
A stochastic process is a collection of random variables $X = X_t;\ t \in T$ such that for each fixed $t \in T$, X_t is a random variable from $(\Omega, \mathcal{F}, \mathbb{P})$ to $(E, \mathcal{G})$. The set Ω is known as the **sample space**, where E is the **state space** of the stochastic process X_t .

The set T can be either discrete, for example the set of positive integers $\mathbb{Z}_+$, or continuous, $T = \mathbb{R}_+$; the state space E will usually be R^d equipped with the σ -algebra of Borel sets.
X_t will represent the random position of our object at time t.

C.1.1 Stochastic Processes Classes

A very important class of continuous-time processes is that of ***Gaussian processes***, which arise in many applications.
A Gaussian process is a stochastic process (a collection of random variables indexed by time or space), such that every finite collection of those random variables has a multivariate normal distribution, i.e. every finite linear combination of them is **normally distributed**.

Definition C.2 (Gaussian Stochastic Process). A time continuous stochastic process $\{X_t; t \in T\}$ is Gaussian if and only if for every finite set of indices $t_1, \ldots, t_k$ in the index set T

$$\mathbf{X}_{t_1,\ldots,t_k} = (X_{t_1}, \ldots, X_{t_k})$$

is a multivariate Gaussian random variable.
That is the same as saying every linear combination of $(X_{t_1}, \ldots, X_{t_k})$ has a univariate normal (or Gaussian) distribution.

The distribution of a Gaussian process is the joint distribution of all those (infinitely many) random variables, and as such, it is a distribution over functions with a continuous domain.

Another class is that of ***Stationary processes***.
In mathematics and statistics, a stationary process is a stochastic process whose unconditional joint probability distribution does not change when shifted in time.

Definition C.3 (Stationary Stochastic Process). Formally, let $\{X_t\}$ be a stochastic process and let $F_X(x_{t_1+\tau}, \ldots, x_{t_n+\tau})$ represent the cumulative distribution function of the unconditional (i.e., with no reference to any particular starting value) joint distribution of $\{X_t\}$ at times $t_1 + \tau, \ldots, t_n + \tau$. Then, $\{X_t\}$ is said to be strictly stationary, strongly stationary or strict-sense stationary if:

$$F_X(x_{t_1+\tau}, \ldots, x_{t_n+\tau}) = F_X(x_{t_1}, \ldots, x_{t_n}) \quad \text{for all } \tau, t_1, \ldots, t_n \in \mathbb{R} \text{ and for all } n \in \mathbb{N}$$

Since τ does not affect $F_X(\cdot)$, F_X is not a function of time.

Consequently, parameters such as mean and variance also do not change over time.

Brownian Motion

Brownian motion is a process with almost surely continuous paths and independent Gaussian increments. A process X_t has independent increments if for every sequence $t_0 < t_1 < \ldots < t_n$, the random variables

$$X_{t_1} - X_{t_0}, X_{t_2} - X_{t_1}, \ldots, X_{t_n} - X_{t_{n-1}}$$

are independent. If furthermore, for every t_1, t_2, $s \in T$ and Borel set $B \subset \mathbb{R}$, we have

$$\mathbb{P}(X_{t_2+s} - Xt_1 + s \in B) = \mathbb{P}(X_{t_2} - X_{t_1} \in B),$$

then the process X_t has stationary independent increments.

Definition C.4. A one-dimensional standard Brownian motion $X(t) : \mathbb{R}^+ \longrightarrow \mathbb{R}$ is a real-valued stochastic process with almost surely (a.s.) continuous paths such that $X(0) = 0$ (with probability 1), it has independent increments, and for every $t > s \geq 0$, the increment $X(t) - X(s)$ has a Gaussian distribution with mean 0 and variance $t - s$, i.e., the density of the random variable $X(t) - X(s)$ is

$$g(x; t, s) = \big(2\pi(t-s)\big)^{-\frac{1}{2}} \exp\bigg(-\frac{x^2}{2(t-s)}\bigg).$$

A standard d-dimensional standard Brownian motion $X(t) : \mathbb{R}^+ \longrightarrow \mathbb{R}^d$ is a vector of d independent one-dimensional Brownian motions:

$$X(t) = (X_1(t), \ldots, X_d(t)),$$

where $X_i(t)$, $i = 1, \ldots, d$ are independent one-dimensional Brownian motions. The density of the Gaussian random vector $X(t) - X(s)$ is thus

$$g(\mathbf{x}; t, s) = \big(2\pi(t-s)\big)^{-\frac{d}{2}} \exp\bigg(-\frac{\|\mathbf{x}\|^2}{2(t-s)}\bigg).$$

Brownian motion is also referred to as a ***Wiener process***.

For computational purposes it is useful to consider discretized Wiener process, where $X(t)$ is specified at discrete t values. We thus set $\delta t = T/N$ for some positive integer N and let X_j denote $X(t_j)$ with $t_j = j\delta t$.
From the definition we can say that

$$X_j = X_{j-1} + dW_j, \quad j = 1, 2, \ldots, N \tag{C.1}$$

where each dW_j is an independent random variable of the form $\sqrt{\delta t}\ N(0, 1)$.

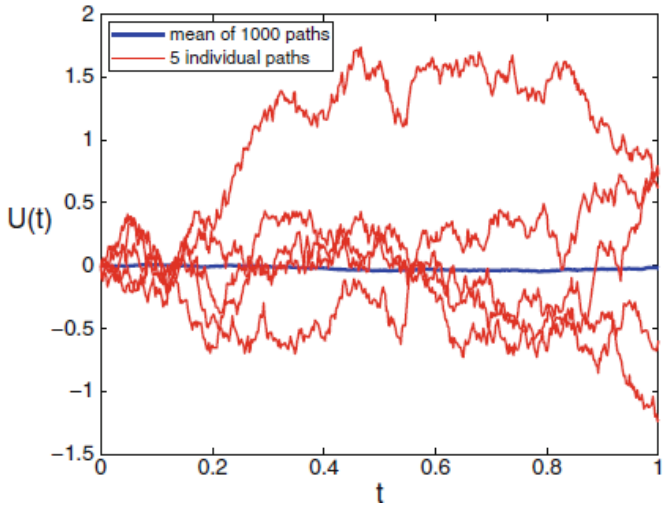


Figure C.1: Brownian sample paths.

C.2 Discretized Brownian paths computation

Given the idea of Brownian motion, we want to compute discretized Brownian paths.
Considering a time interval $[0, T]$, N discretization step and n number of particles, the MATLAB function *wiener_process* performs the Wiener Process corresponding to each considered particle.
Here the diffusion coefficient D is chosen equal to 1 and we use the MATLAB function *normrnd* to generate a random numbers from the normal distribution with mean $\mu = 0$ and standard deviation $sd = 1$.
The initial point for each Wiener process is 0, so $X(0) = 0$.

```
%T=1 %N=100 %n=1000
function [X]=wiener_process(T,N,n)
%This function creates the Wiener Process
% (Standard Brownian Motion )
%input: T= time period;
%       N= discrtizion step;
%       n= particles number;
%output:X= Wiener process;

dt=T/N;
t=[0:dt:T]; %instant of time vector
mu=0;
```

```
sd=1;
D=1; %diffusion coefficient
h=sqrt(2*D);
X(:,1)=zeros(n,1);
%normrnd: generates a random number from the normal
%          distribution with mean parameter mu and
%          standard deviation parameter sd
%          [N,1] indicates the size of each dimension
for i= 1:N
X(:,i+1)=X(:,i)+h*sqrt(dt)*normrnd(mu,sd,[n,1]);
end
%now we plot the Wiener process for each particle
plot(t,X)
xlabel('Time'), ylabel('Process State')
title('Wiener Process')
end
```

Varying the number of particles n, we can have different numbers of paths.

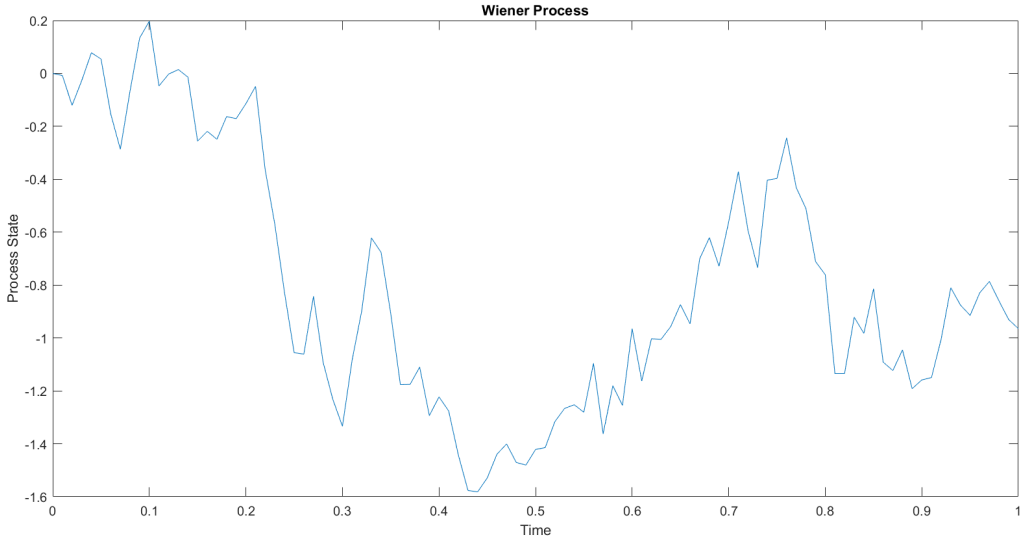


Figure C.2: Trajectory of one particle with T=1 and N=100.

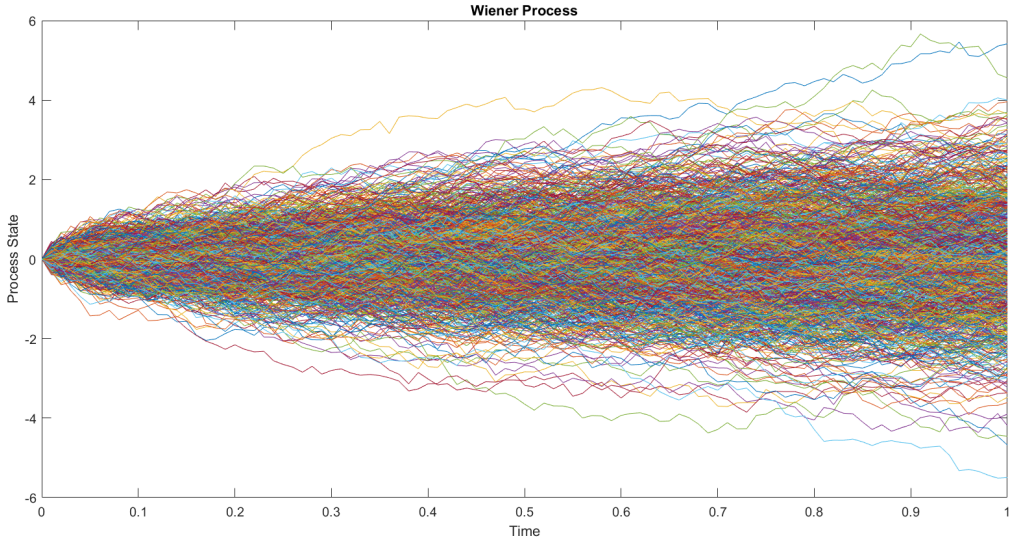


Figure C.3: Trajectories of $n = 1000$ particles with T=1 and N=100.

C.2.1 Distribution of the Wiener Process

In this section we want to study the distribution of the Wiener Process and verify if it can be compared with a Gaussian distribution.

In order to do this comparison, the *hist_tot* MATLAB function first create the Wiener Process for $n = 1000$ particles and the corresponding histogram.

The trend of the histogram was then considered and subsequently it has been compared with the Gaussian trend.

```
%T=1 %N=100 %n=10000
function [X]=hist_tot(T,N,n)
% This function looks at the density distribution of the
% standard Browian motion and verifies that it has a
% Gaussian distribution.

%input: T= time period;
%       N= discrtizion step;
%       n= particles number;
%output:X= Wiener process;

```

```
%Wiener Process
[X]=wiener_process(T,N,n);

figure(1)
n_in=100;
[nn,y] = hist(X(:,end),n_in); % hist matlab function
n_norm = (nn ./length(X(:,end))) ./(y(2)-y(1));
                              % normalize hist func
bar(y,n_norm); hold ('on');
plot(y,n_norm,'r');
xlabel('x'), ylabel('Distribution')
title('Histogram of the Wiener Process')

figure(2);
semilogy(y,n_norm,'r'); hold('on');
%plot(y,n_norm,'r'); hold('on');
pd = fitdist(X(:,end),'Normal'); %creating the gaussian function
f = pdf(pd, y);
c1=[0.6350 0.0780 0.1840];
scatter(y,f,10,c1, 'filled')
xlabel('x'), ylabel('Distribution')
title('Histogram of the Wiener Process')

figure(3)
bar(y,n_norm); hold('on');
pd = fitdist(X(:,end),'Normal');
f = pdf(pd, y);
c=[0 0.4470 0.7410];
scatter(y,f,10,c1, 'filled')
set(gca,'yscale','log')
xlabel('x'), ylabel('Distribution')
title('Histogram of the Wiener Process')

figure(3);
plot(y,n_norm,'r'); hold('on');
pd = fitdist(X(:,end),'Normal'); %creating the gaussian function
f = pdf(pd, y);
c1=[0.6350 0.0780 0.1840];
scatter(y,f,10,c1, 'filled')
xlabel('x'), ylabel('Distribution')
title('Histogram of the Wiener Process')

figure(5)
```

```
55 bar(y,n_norm); hold('on');
   pd = fitdist(X(:,end),'Normal');
57 f = pdf(pd, y);
   c=[0 0.4470 0.7410];
59 scatter(y,f,10,c1, 'filled')
   xlabel('x'), ylabel('Distribution')
61 title('Histogram␣of␣the␣Wiener␣Process')
   end
```

Figures C.45, C.46 and C.47 show the comparison using a logarithmic scale.

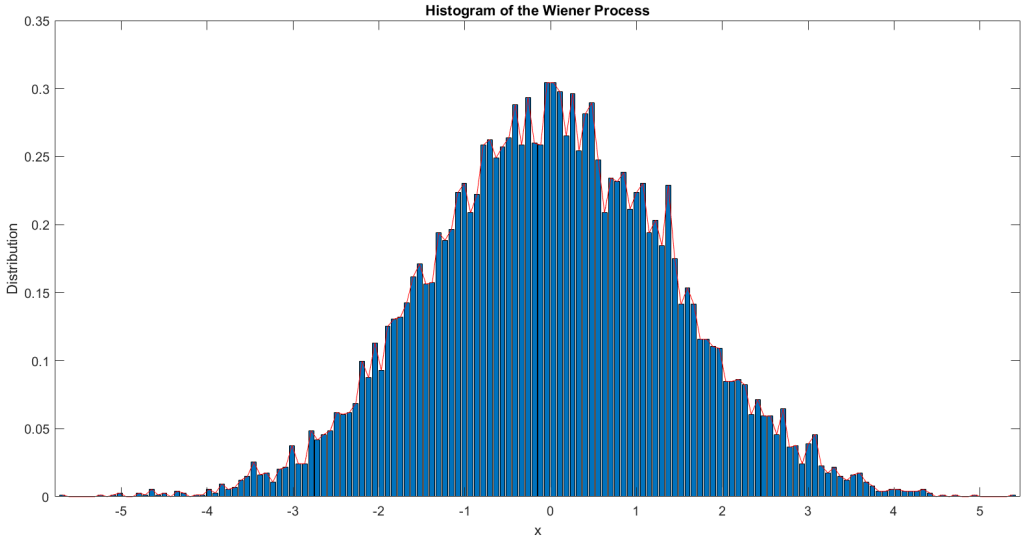


Figure C.4: Histogram of $X(t)$ and its distribution with T=1, N=100, n=10000.

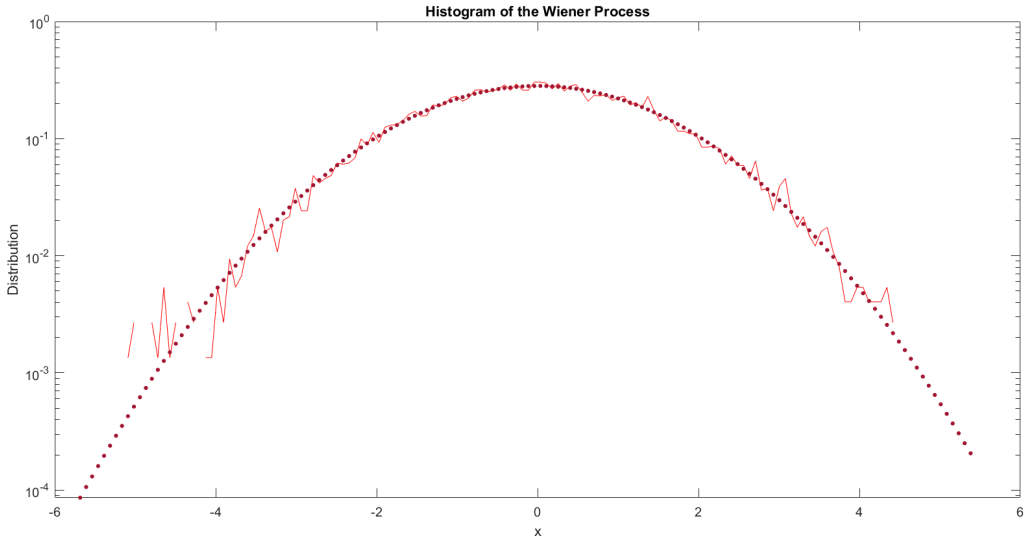


Figure C.5: Comparison between the histogram distribution trend and the Gaussian one with T=1, N=100, n=10000.

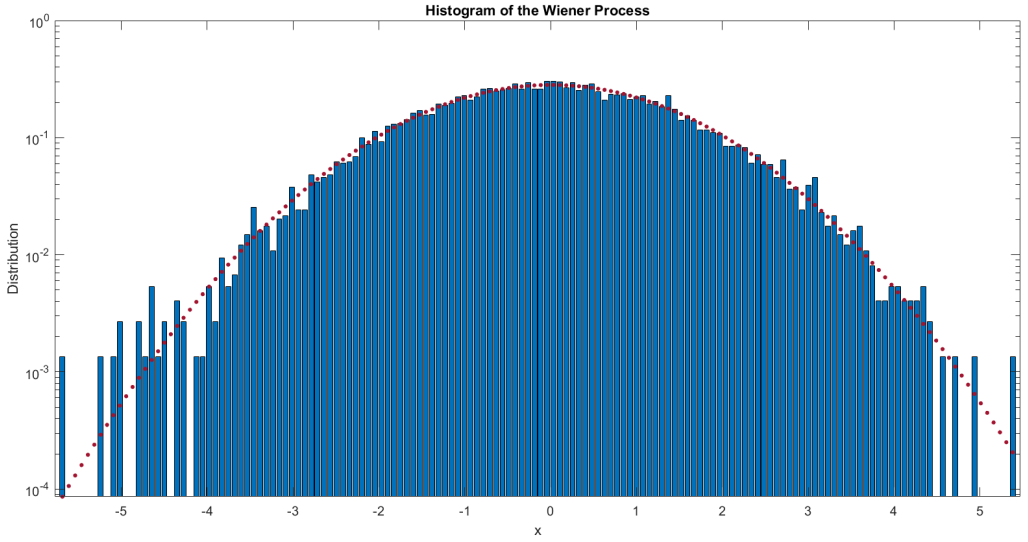


Figure C.6: Gaussian distribution on $X(t)$ histogram with T=1, N=100, n=10000.

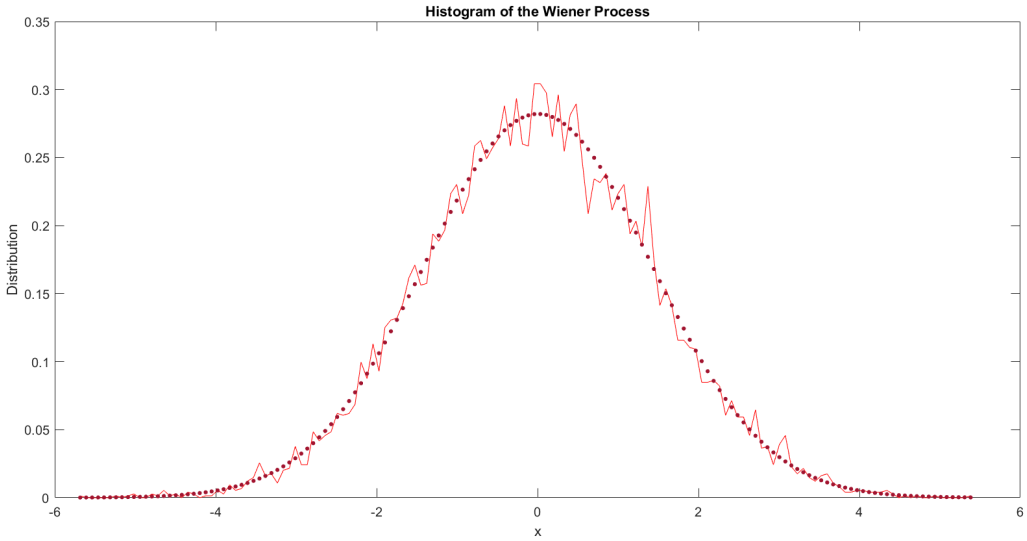


Figure C.7: Histogram of $X(t)$ and its distribution with T=1, N=100, n=10000.

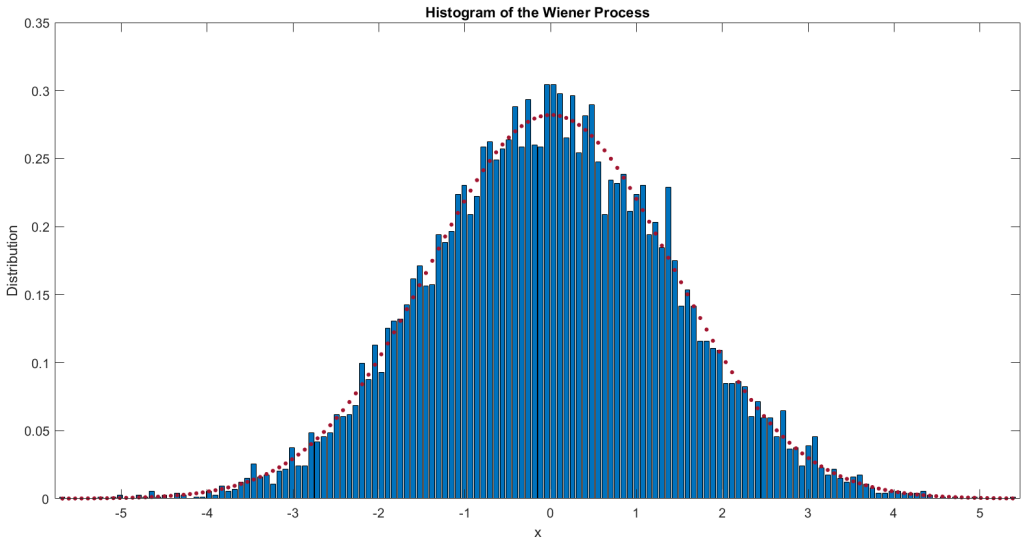


Figure C.8: Comparison between the histogram distribution trend and the Gaussian one with T=1, N=100, n=10000.

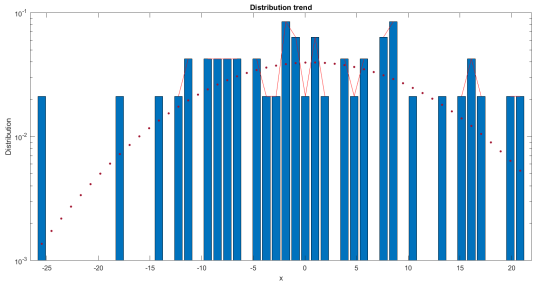
Distribution of the Wiener Process with different n and T

Now we want to verify the similitude between the histogram distribution and the Gaussian one, changing both the time T and the number of particles n. In this way it is possible to notice how the empirical distribution of the histogram resembles the Gaussian one as time and particles increase.

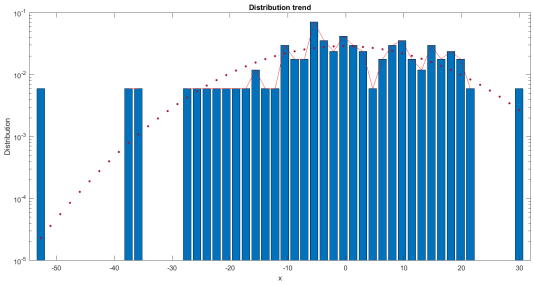
The *histograms* MATLAB function performs an unique plot with the histogram, the empirical distribution and the Gaussian distribution.
Recalling this function changing the input arguments, it is possible to obtain different plots in order to highlight the behaviour of the Wiener process.

```
%N=500
function [X,f]=histograms(T,N,n)
% This function looks at the density distribution of the
% Wiener Process and verifies that it has a Gaussian
% distribution changing times and the particle number.
%input: T= time period;
%       N= discrtizion step;
%       n= particles number;
%output:X= Wiener Process;
%       f= density distribution function

%Wiener Process
[X]=wiener_process(T,N,n);

%histogram with empirical and gaussian distribution
[nn,y] = hist(X(:,end),50);
n_norm = (nn ./length(X(:,end))) ./(y(2)-y(1));
bar(y,n_norm); hold('on');
set(gca, 'YScale', 'log');
plot(y,n_norm,'r'); hold('on');
pd = fitdist(X(:,end),'Normal');
f = pdf(pd, y);
c=[0.6350 0.0780 0.1840];
scatter(y,f,10,c, 'filled')

xlabel('x'), ylabel('Distribution')
title('Distribution trend')
end


```

```
figure(1); n1=50; T1=50; [X,f]=histograms(T1,N,n1);
figure(2); n2=100; T2=100; [X,f]=histograms(T2,N,n2);
figure(3); n3=250; T3=200; [X,f]=histograms(T3,N,n3);
figure(4); n4=500; T4=500; [X,f]=histograms(T4,N,n4);
figure(5); n5=800; T5=1000; [X,f]=histograms(T5,N,n5);
figure(6); n6=1000; T6=10000; [X,f]=histograms(T6,N,n6);
```

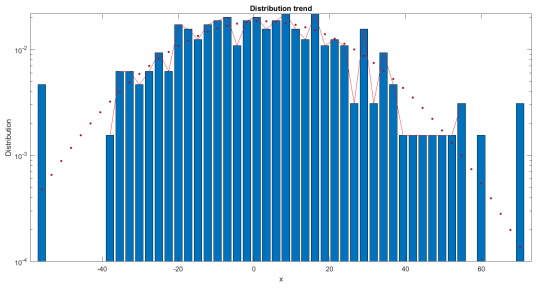


N=500, T=50, n=50

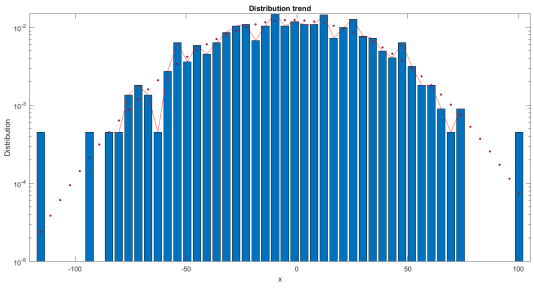


N=500, T=100, n=100

Figure C.9

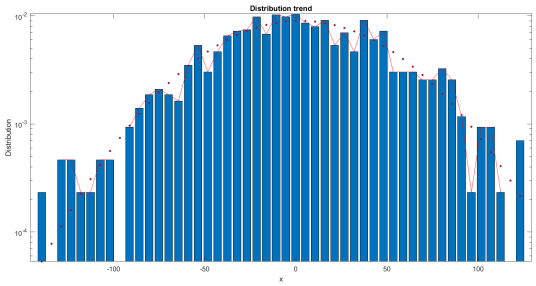


N=500, T=250, n=200

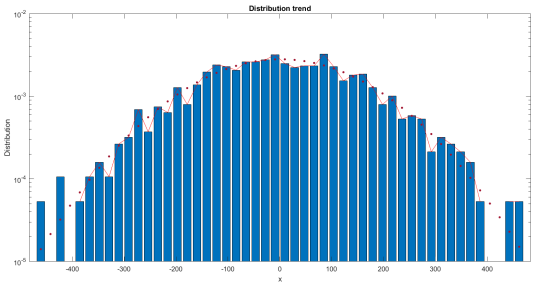


N=500, T=500, n=500

Figure C.10



N=500, T=1000, n=800

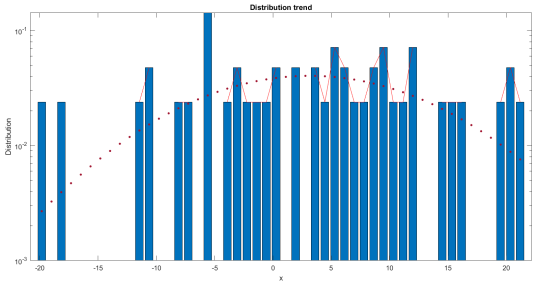


N=500, T=10000, n=1000

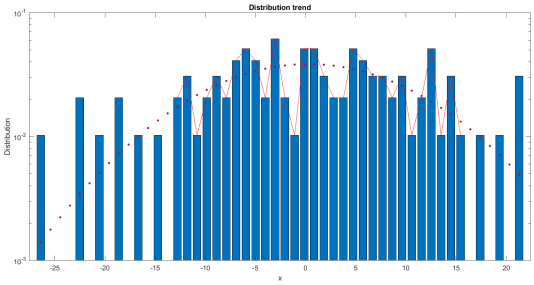
Figure C.11

If we fix the time T and we change the particles number n, it is possible to see how the fit gets better.

```
%T=50; %N=500;
figure(1); n1=50; [X,f]=histograms(T,N,n1);
figure(2); n2=100; [X,f]=histograms(T,N,n2);
figure(3); n3=250; [X,f]=histograms(T,N,n3);
figure(4); n4=500; [X,f]=histograms(T,N,n4);
figure(5); n5=800; [X,f]=histograms(T,N,n5);
figure(6); n6=1000; [X,f]=histograms(T,N,n6);
```

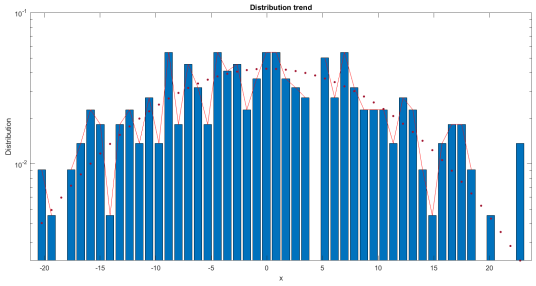


T=50, N=500, n=50

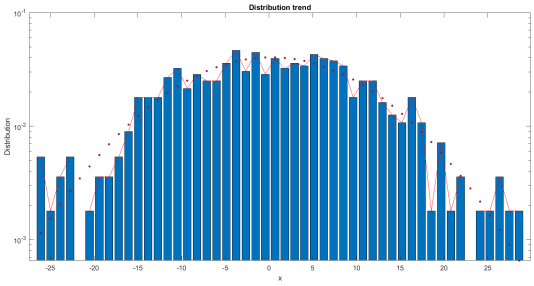


T=50, N=500, n=100

Figure C.12

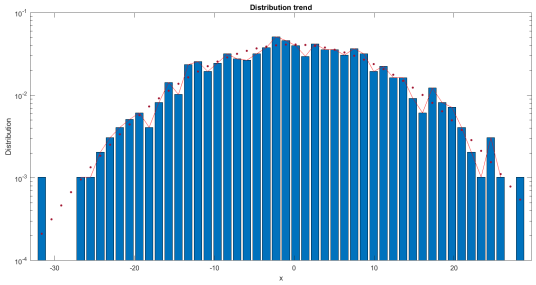


T=50, N=500, n=200

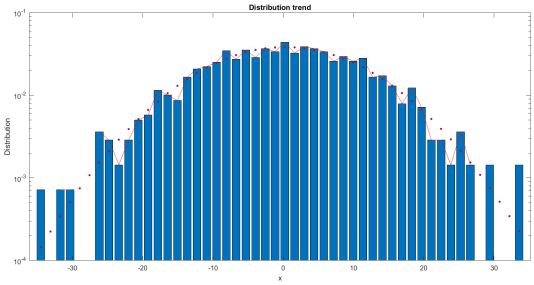


T=50, N=500, n=500

Figure C.13



T=50, N=500, n=800



T=50, N=500, n=1000

Figure C.14

C.2.2 Variance

The aim of this section is to show the variance behaviour with respect to time: we want calculate the variance σ^2 and show that it has a proportional trend with respect to time t.

The variance of the process $X(t)$ was calculated by the MATLAB function *var* and then it was compared with the linear trend $V = 2*D*t$ because theoretically we know it should be like that.

```
%T=1; N=100; n=7000;
function [X] = variance_D(T,N,n,D)
% This function displays the variance of the Wiener Process
% varying the diffusion coefficient D and compare its
% trend with the linear one.
%input: T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%output:X= Wiener process;

%construction of th Wiener process
dt=T/N;
t=[0:dt:T];
mu=0;
sd=1;
h=sqrt(2*D*dt);
X(:,1)=zeros(n,1);
for i= 1:N
X(:,i+1)=X(:,i)+h*normrnd(mu,sd,[n,1]);
end

V=var(X(:,:),1);
t = [0:dt:T];
VV=2*D*t;
plot(t,V)
hold on
plot(t,VV, 'Linewidth', 1)
xlabel('Time'), ylabel('Variance')
title('Variance Trend')
end
```

Plotting the variance with respect to time, we can see that the trend is quite linear. There is some noise due to the numerical error but we can notice that increasing the number of particles n it is possible to obtain a better plot.
This is showed in the following figures.

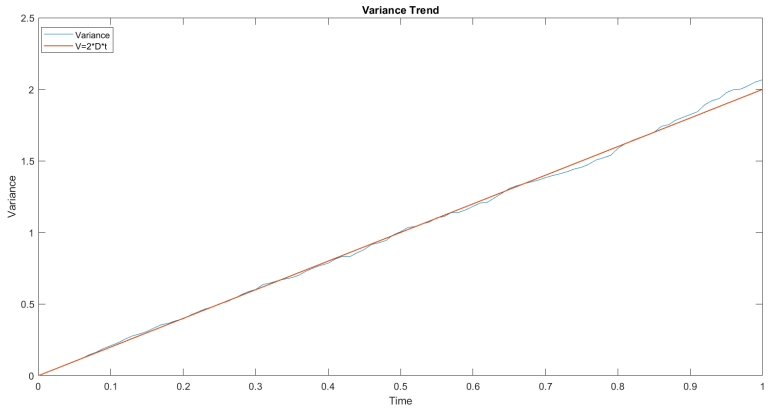


Figure C.15: $T = 1$, $N = 500$, $D = 1$, $n = 1000$

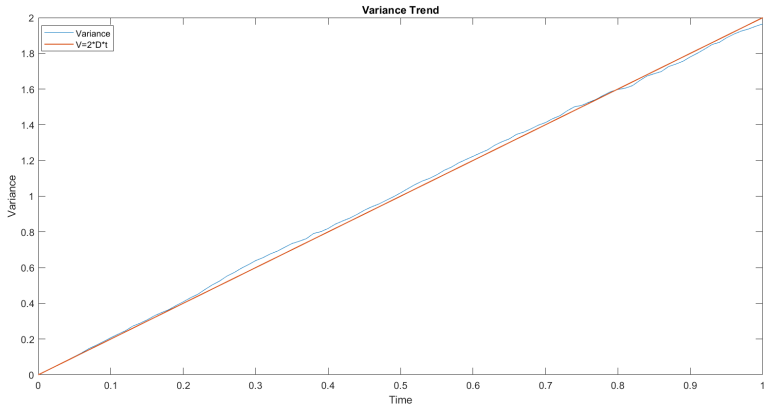


Figure C.16: $T = 1$, $N = 500$, $D = 1$, $n = 3000$

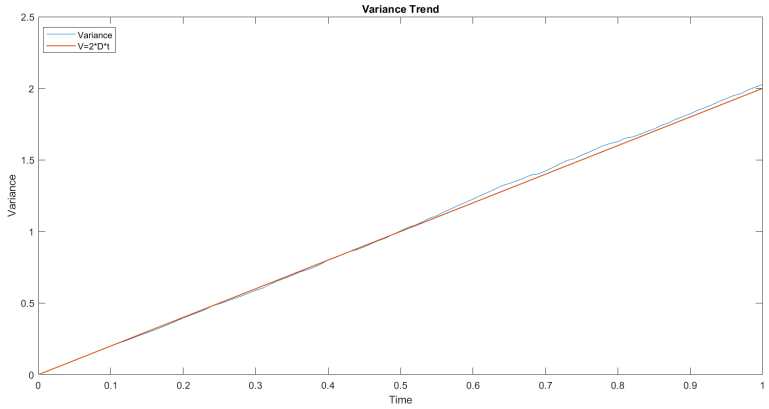


Figure C.17: $T = 1$, $N = 500$, $D = 1$, $n = 5000$

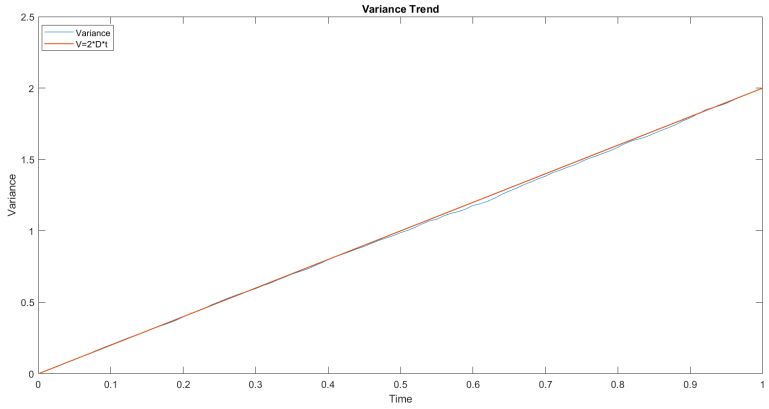


Figure C.18: $T = 1$, $N = 500$, $D = 1$, $n = 7000$

Changing the diffusion coefficient D it is possible to obtain a plot as the one in figure C.19.

```
%T=1; N=100; n=7000;
figure(1);
hold ('on')
D1=0.1; [X] = variance_D(T,N,n,D1);
D2=0.5; [X] = variance_D(T,N,n,D2);
D3=1; [X] = variance_D(T,N,n,D3);
D4=3; [X] = variance_D(T,N,n,D4);
D5=7; [X] = variance_D(T,N,n,D5);
D6=10; [X] = variance_D(T,N,n,D6);
D7=20; [X] = variance_D(T,N,n,D7);
```

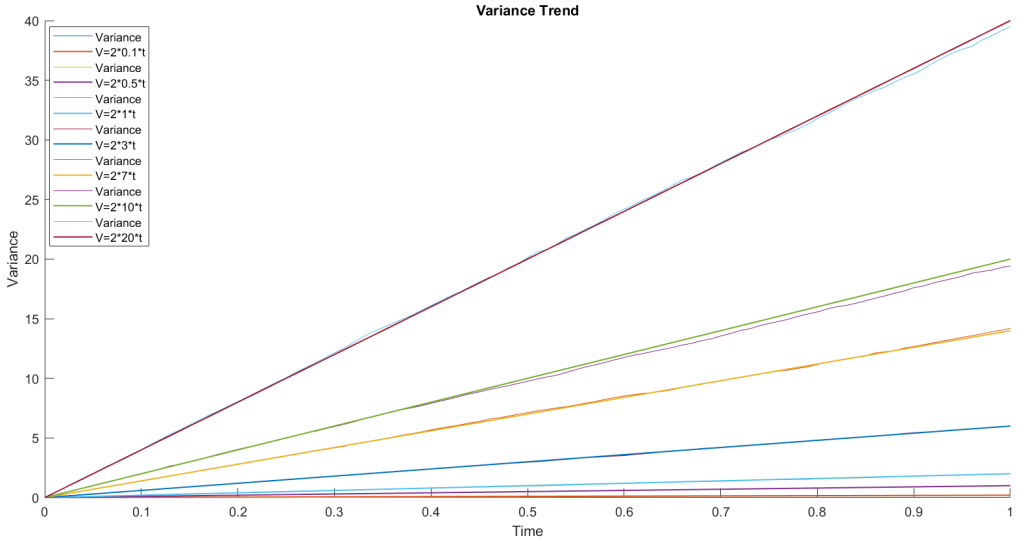


Figure C.19: $T = 1$, $N = 500$, $n = 7000$, $D1 = 0.1$, $D2 = 0.5$, $D3 = 1$, $D4 = 3$, $D5 = 7$, $D6 = 10$, $D7 = 20$

Gaussian distributions

Using the scale law obtained from the variance calculation we want to verify that, as time varies, all distributions tend to the same Gaussian curve.

In the MATLAB function *gaussian_tt* first the Wiener process is built and then it is considered with respect to the standard deviation σ as $\frac{X(t)}{\sigma(t)}$.

After that, it is possible to consider the Gaussian distribution and its trend, varying the time T, is showed in figure C.20.

```
function [X,f,y]=gaussian_tt(T,N,n,D)
%The gaussian function allows us to see the density distribution
%input: T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%output:X= Wiener process;
%       f= density distribution function
%       y= histogram columns centers

%construction of th Wiener process
```

```
dt=T/N;
t=[0:dt:T];
mu=0;
sd=1;
h=sqrt(2*D);
X(:,1)=zeros(n,1);
for i= 1:N
X(:,i+1)=X(:,i)+h*sqrt(dt)*normrnd(mu,sd,[n,1]);
end

sigma=sqrt(var(X(:,:),1));
X=X/sigma;

n_in=100;
[nn,y] = hist(X,n_in);
n_norm = (nn ./length(X(:,end))) ./(y(2)-y(1));
pd = fitdist(X,'Normal');
f = pdf(pd, y);

c=[0.4660 0.6740 0.1880];
scatter(y,n_norm,5,c, 'filled')

xlabel('X(t)/sigma(t)'), ylabel('Distribution sigma(t)')
title('Gaussian Trends')
end

figure(1);
hold on
T1=0.1; [X,f,y]=gaussian_tt(T1,N,n,D);
T2=1; [X,f,y]=gaussian_tt(T2,N,n,D);
T3=10; [X,f,y]=gaussian_tt(T3,N,n,D);
T4=100; [X,f,y]=gaussian_tt(T4,N,n,D);
T5=1000; [X,f,y]=gaussian_tt(T5,N,n,D);
T6=100000; [X,f,y]=gaussian_tt(T6,N,n,D);
T7=10000000; [X,f,y]=gaussian_tt(T7,N,n,D);
plot(y,f,'Linewidth',0.7)
```

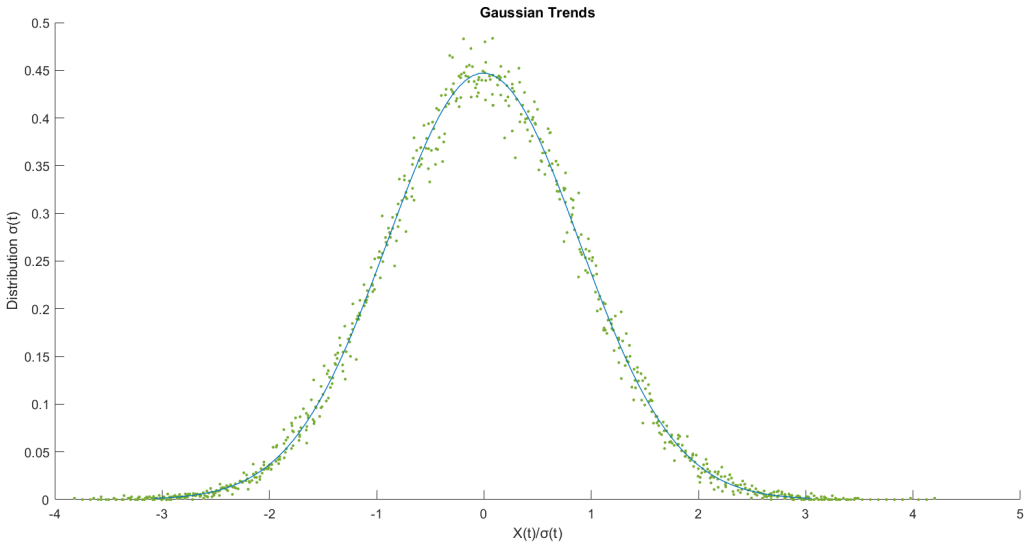


Figure C.20: N=500, n=7000, D=1, T1=0.1, T2=1, T3=10, T4=100, T5=1000, T6=100000, T7=10000000.

C.3 Ornstein-Uhlenbeck Process

The classical Gaussian Ornstein–Uhlenbeck (OU) process was first introduced by Uhlenbeck and Ornstein (1930) in physics, and later became popular in finance, economics and many other fields.
Its original application in physics was as a model for the velocity of a massive Brownian particle under the influence of friction.

C.3.1 General facts

The Ornstein–Uhlenbeck process was introduced as a model for the velocity of a Brownian particle, it is a stationary Gauss–Markov process, which means that it is a Gaussian process, a Markov process, and is temporally homogeneous. In fact, it is the only nontrivial process that satisfies these three conditions, up to allowing linear transformations of the space and time variables. Over time, the process tends to drift towards its mean function: such a process is called *mean-reverting*.

The process can be considered to be a modification of the Wiener process, in which the properties of the process have been changed so that there is a tendency of the walk to move back towards a central location, with a greater

attraction when the process is further away from the center.

Roughly speaking, a Markov process is a stochastic process that retains no memory of where it has been in the past: only the current state of a Markov process can influence where it will go next. A bit more precisely, a Markov process is a stochastic process whose past and future are statistically independent, conditioned on its present state. Perhaps the simplest example of a Markov process is a random walk in one dimension.

Brownian motion and the Ornstein–Uhlenbeck process are examples of a diffusion process: a continuous-time Markov process with continuous paths.

Definition

In many stochastic processes that appear in applications, their statistics remain invariant under time translations. Such stochastic processes are called stationary. It is possible to develop a quite general theory of stochastic processes that enjoy this symmetry property.
It is useful to distinguish between stochastic processes for which all FDDs are translation-invariant (*strictly stationary processes*) and processes for which this translation invariance holds only for the first two moments (*weakly stationary processes*).

Definition C.5 (**Strictly Stationary Processes**). A stochastic process is called (strictly) stationary if all FDDs are invariant under time translation: for every integer k and for all times $t_i \in T$, the distribution of $(X(t_1), X(t_2), \ldots, X(t_k))$ is equal to that of $(X(s+t_1), X(s+t_2), \ldots, X(s+t_k))$ for every s such that $s + t_i \in T$ for all $i \in 1, \ldots, k$. In other words,

$$\mathbb{P}(X_{t_1+s} \in A_1, Xt_2 + s \in A_2 \ldots X_{t_k+s} \in A_k) = \mathbb{P}(X_{t_1} \in A_1, X_{t_2} \in A_2 \ldots X_{t_k} \in A_k), \forall s \in T \tag{C.2}$$

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. Let X_t, $t \in T$ (with $T = \mathbb{R}$ or $\mathbb{Z}$), be a realvalued random process on this probability space with finite second moment $\mathbb{E}\|X_t\|^2 < +\infty$ (i.e., $X_t \in L^2(\Omega, \mathbb{P})$ for all $t \in T$). Assume that it is strictly stationary. Then

$$\mathbb{E}(X_{t+s}) = \mathbb{E}X_t, \quad s \in T, \tag{C.3}$$

from which we conclude that EXt is constant, and

$$\mathbb{E}((X_{t_1+s} - \mu)(X_{t_2+s} - \mu)) = \mathbb{E}((X_{t_1} - \mu)(X_{t_2} - \mu)), \quad s \in T, \tag{C.4}$$

implies that the *covariance function* depends on the difference between the two times t and s: $C(t,s) = C(t-s)$.
This motivates the following definition.

Definition C.6. A stochastic process $X_t \in L^2$ is called second-order stationary, widesense stationary, or weakly stationary if the first moment $\mathbb{E}X_t$ is a constant and the covariance function $\mathbb{E}(X_t - \mu)(X_s - \mu)$ depends only on the difference t-s:

$$\mathbb{E}X_t = \mu, \quad \mathbb{E}((X_t - \mu)(X_s - \mu)) = C(t-s). \tag{C.5}$$

The constant μ is the expectation of the process X_t. The function $C(t)$ is the *covariance* (sometimes also called autocovariance) or the *autocorrelation function* of the X_t. Notice that $C(t) = \mathbb{E}(X_t X_0)$, whereas $C(0) = \mathbb{E}X_t^2$, which is finite, by assumption. Since we have assumed that X_t is a real valued process, we have that $C(t) = C(-t)$, $t \in \mathbb{R}$.

A strictly stationary process with finite second moment is also stationary in the wide sense. The converse is not true, in general.
It is true, however, for Gaussian processes: since the first two moments of a Gaussian process are sufficient for a complete characterization of the process, a Gaussian stochastic process is strictly stationary if and only if it is weakly stationary.

The covariance function of a second-order stationary process is a *nonnegative* definite function and it enables us to associate a timescale to X_t, the *correlation time* τ_{cor}:

$$\tau_{cor} = \frac{1}{C_0}\int_0^\infty C(\tau)d\tau = \frac{1}{\mathbb{E}(X_0^2)}\int_0^\infty \mathbb{E}(X_\tau X_0)d\tau. \tag{C.6}$$

The slower the decay of the correlation function, the larger the correlation time is. Note that when the correlations do not decay sufficiently fast, so that $C(t)$ is not integrable, then the correlation time will be infinite.

If we consider a mean-zero second-order stationary process with correlation function

$$C(t) = C(0)e^{-\alpha\|t\|}, \quad with\ \alpha > 0. \tag{C.7}$$

We have that $C(0) = \frac{D}{\alpha}$ and the correlation time is

$$\tau_{cor} = \int_0^\infty e^{-\alpha t}dt = \alpha^{-1} \tag{C.8}$$

A real-valued Gaussian stationary process defined on R with correlation function given by (C.7) is called a stationary *Ornstein–Uhlenbeck process.*

The stationary Ornstein–Uhlenbeck process can be defined through the Brownian motion via a time change. It is a stochastic process that satisfies the following stochastic differential equation:

$$dX_t = \kappa(\theta - X_t)dt + \sigma dW_t \tag{C.9}$$

where W_t is the Wiener Process on $t \in [0, \infty)$ and the constant parameters are:

- $\kappa > 0$ is the rate of mean reversion;
- θ is the long-term mean of the process;
- $\sigma > 0$ is the diffusion coefficient.

The solution to the stochastic differential equation (C.9) defining the Ornstein-Uhlenbeck process is, for any $0 \leq s \leq t$, is

$$X_t = \theta + (X_s - \theta)e^{-\kappa(t-s)} + \sigma \int_s^t e^{-\kappa(t-u)} dW_u \tag{C.10}$$

where the integral on the right is the Itô integral.

For any fixed s and t, the random variable X_t, conditional upon X_s, is normally distributed with

$$mean = \theta + (X_s - \theta)e^{-\kappa(t-s)}, \quad variance = \frac{\sigma^2}{2\kappa}(1 - e^{-2\kappa(t-s)}) \tag{C.11}$$

It is possible to notice that the Ornstein-Uhlenbeck process is a time-homogeneous Ito diffusion.

C.3.2 Process construction

Let us consider a Wiener Process with no drift and constant diffusion coefficient σ:

$$X_n = X_{n-1} + \sqrt{2\sigma} dW_n, \quad X(0) = x \tag{C.12}$$

where the initial condition X_0 can be either deterministic or random.
Adding a restoring force to (C.12), we obtain the Ornstein-Uhlenbeck process

in the following form:

$$X_n = X_{n-1} - \frac{X_{n-1}}{\tau_{cor}} dt + \sqrt{2\sigma} dW_n, \quad X(0) = x. \tag{C.13}$$

Considering a time interval $[0, T]$, N discretization step and n number of particles,the MATLAB function *ornstein_uhlenbeck* performs the Ornstein-Uhlenbeck Process corresponding to each considered particle.
Here the diffusion coefficient D and the correlation time τ are chosen equal to 1 and we use the MATLAB function normrnd to generate a random numbers from the normal distribution with mean $\mu = 0$ and standard deviation $sd = 1$.
The initial point for each Ornstein-Uhlenbeck process is random.

```
%X0=randn(n,N); T=1; N=500; D=1; tau=1;

function [X]=ornstein_uhlenbeck (X0,T,N,n,D,tau)
%This function creates the Ornstein-Uhlenbeck Process
%input: X0=initial point;
%       T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%       tau=correlation time;
%output:X= Ornstein-Uhlenbeck process;

dt=T/N;
t=(0:dt:T); %instant of time vector
mu=0; %mean
sd=1; %standard deviation
h=sqrt(2*D);
X=zeros(n,N);
X(:,1)=X0(:,1);
%normrnd: generates a random number from the normal
%         distribution with mean parameter mu and
%         standard deviation parameter sd [N,1]
%         indicates the size of each dimension
for i= 1:N
X(:,i+1)=X(:,i)-((X(:,i)/tau)*dt)+h*sqrt(dt)*normrnd(mu,sd,[n,1]);
end
plot(t,X)
xlabel('Time'), ylabel('Process State')
title('Ornstein-Uhlenbeck Process')
```

```
end
```

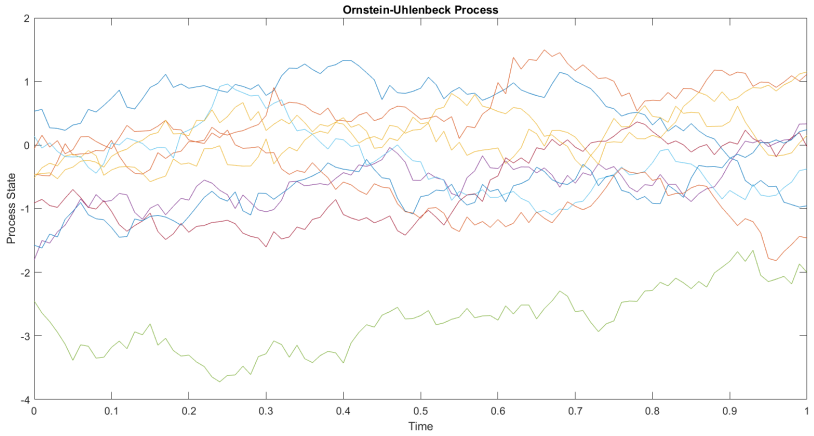


Figure C.21: Trajectory of n=10 particle with T=1, N=500, D=1, τ_{cor}=1.

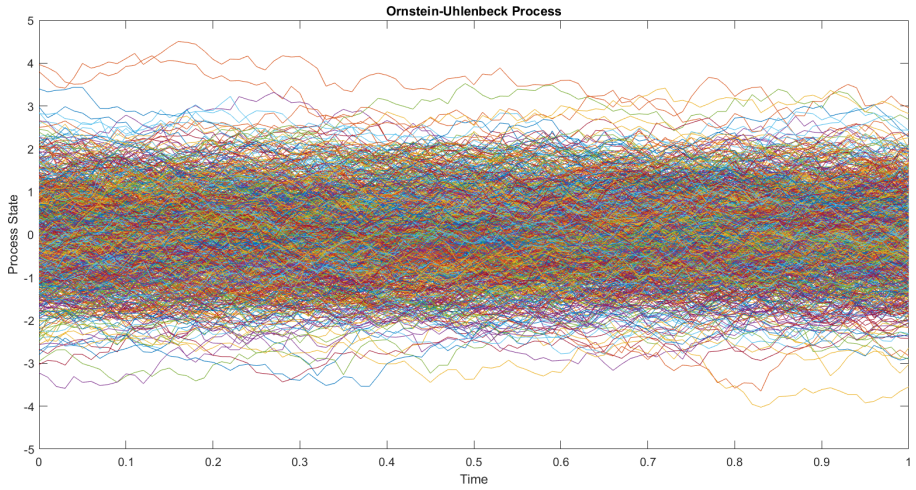


Figure C.22: Trajectory of n=1000 particle with T=1, N=500, D=1, τ_{cor}=1.

It is possible to verify that if we consider $\tau_{cor} = \infty$ the Ornstein-Uhlendeck process become a Wiener process; this is shown in figure C.23.

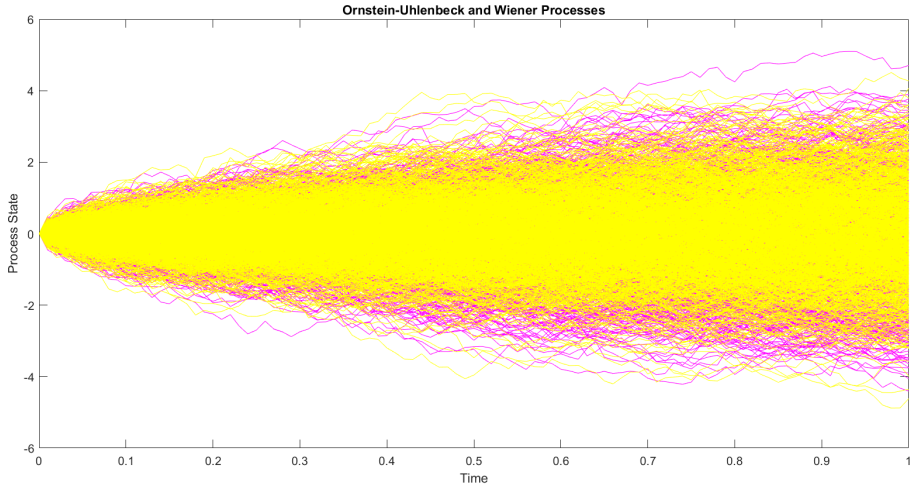


Figure C.23: In pink the Ornstein-Uhlendeck process trajectories of n=1000 particles and in yellow their Wiener process trajectories.
X_0=0, T=1, N=500, n=1000, D=1, τ_{cor}=1.

C.3.3 Distribution of the Ornstein-Uhlenbeck Process

In this section we want to study the distribution of the Ornstein-Uhlenbeck process and verify if it can be compared with a Gaussian distribution.

As we did for the Wiener Process, in order to do this comparison we use the *hist_tot_ou* MATLAB function which first create the Ornstein-Uhlenbeck Process for $n = 10000$ particles and then the corresponding histogram.

The trend of the histogram was then considered and subsequently it has been compared with the Gaussian trend.

```
%X0=randn; T=10; N=500; n=10000; D=1; tau=1;

function [X]=hist_tot_ou(X0,T,N,n,D,tau)
% This function looks at the density distribution of the
% Ornstein-Uhlenbeck process and verifies
% that it has a Gaussian distribution.
```

```
%input: X0=initial point;
%        T= time period;
%        N= discrtizion step;
%        n= particles number;
%        D= diffusion coefficient;
%        tau=correlation time;
%output:X= Ornstein-Uhlenbeck process;

%Ornstein-Uhlenbeck process construction
[X]=ornstein_uhlenbeck (X0,T,N,n,D,tau);

figure(1)
n_bins=150;
[nn,y] = hist(X(:,end),n_bins); % hist matlab function
n_norm = (nn ./length(X(:,end))) ./(y(2)-y(1));
                                    %normalize hist function
bar(y,n_norm); hold ('on');
plot(y,n_norm,'r');
xlabel('x'), ylabel('Distribution')
title('Histogram␣of␣the␣Ornstein-Uhlenbeck␣process')

figure(2);
semilogy(y,n_norm,'r'); hold('on');
%plot(y,n_norm,'r'); hold('on');
pd = fitdist(X(:,end),'Normal'); %creating the gaussian function
f = pdf(pd, y);
c1=[0.6350 0.0780 0.1840];
scatter(y,f,10,c1, 'filled')
xlabel('x'), ylabel('Distribution')
title('Histogram␣of␣the␣Ornstein-Uhlenbeck␣process')

figure(3)
bar(y,n_norm); hold('on');
pd = fitdist(X(:,end),'Normal');
f = pdf(pd, y);
c=[0 0.4470 0.7410];
scatter(y,f,10,c1, 'filled')
set(gca,'yscale','log')
xlabel('x'), ylabel('Distribution')
title('Histogram␣of␣the␣Ornstein-Uhlenbeck␣process')

figure(4);
```

```
plot(y,n_norm,'r'); hold('on');
pd = fitdist(X(:,end),'Normal'); %creating the gaussian function
f = pdf(pd, y);
c1=[0.6350 0.0780 0.1840];
scatter(y,f,10,c1, 'filled')
xlabel('x'), ylabel('Distribution')
title('Histogram of the Ornstein-Uhlenbeck process')

figure(5)
bar(y,n_norm); hold('on');
pd = fitdist(X(:,end),'Normal');
f = pdf(pd, y);
c=[0 0.4470 0.7410];
scatter(y,f,10,c1, 'filled')
xlabel('x'), ylabel('Distribution')
title('Histogram of the Ornstein-Uhlenbeck process')
end
```

Figures C.24, C.25 and C.26 show the comparison using a logarithmic scale.

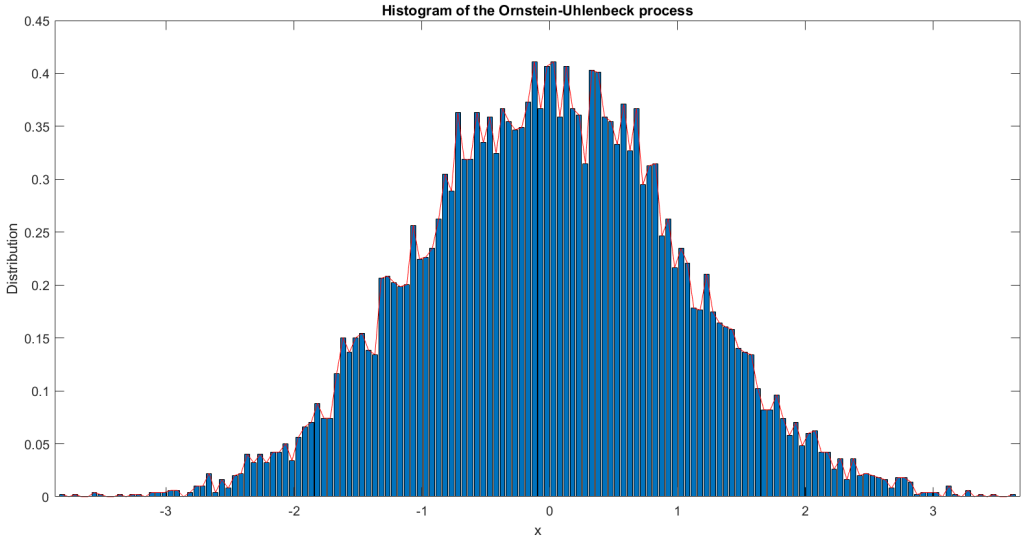


Figure C.24: Histogram of $X(t)$ and its distribution with X_0=randn, T=10, N=500, n=10000, D=1, τ_{cor}=1.

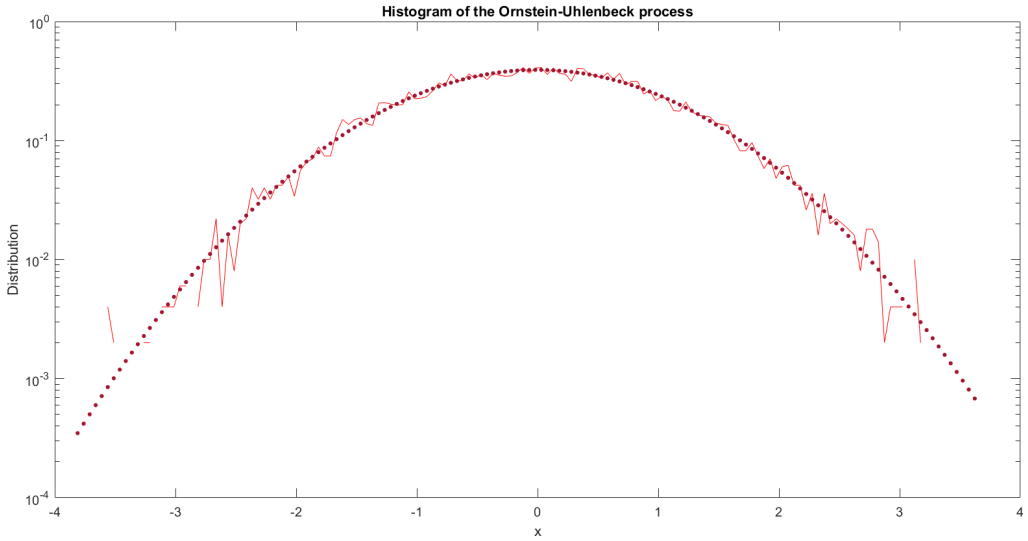


Figure C.25: Comparison between the histogram distribution trend and the Gaussian one with X_0=randn, T=10, N=500, n=10000, D=1, τ_{cor}=1.

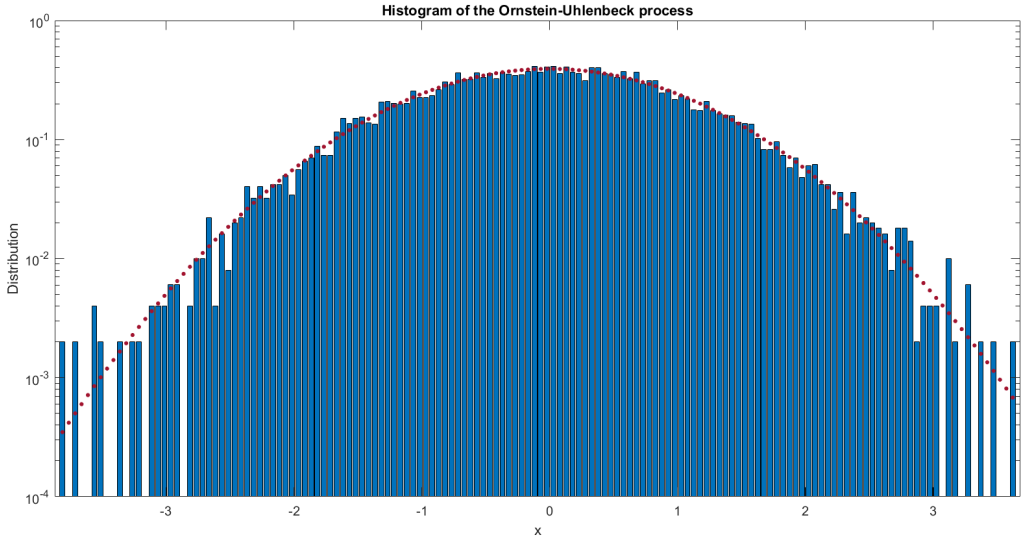


Figure C.26: Gaussian distribution on $X(t)$ histogram with X_0=randn, T=10, N=500, n=10000, D=1, τ_{cor}=1.

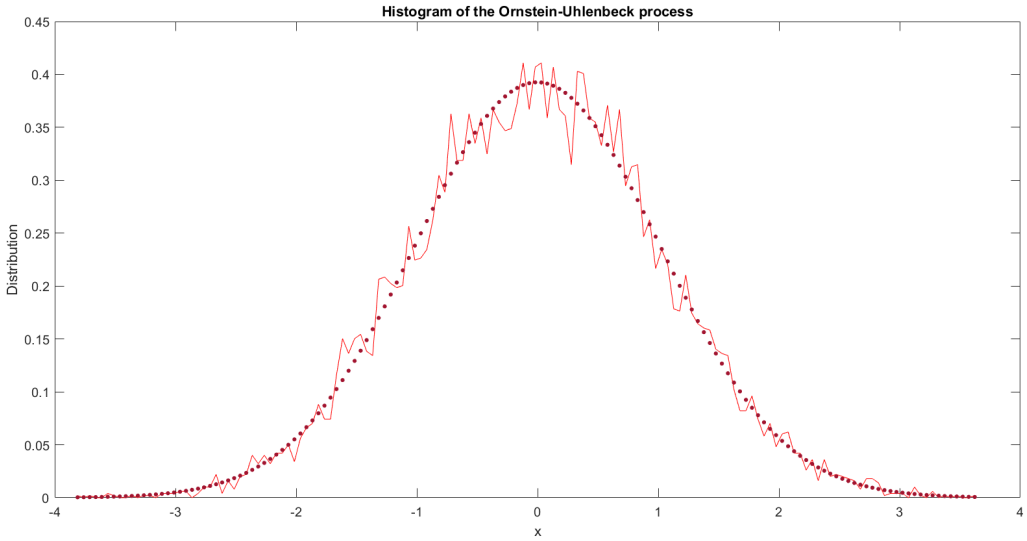


Figure C.27: Histogram of $X(t)$ and its distribution with X_0=randn, T=10, N=500, n=10000, D=1, τ_{cor}=1.

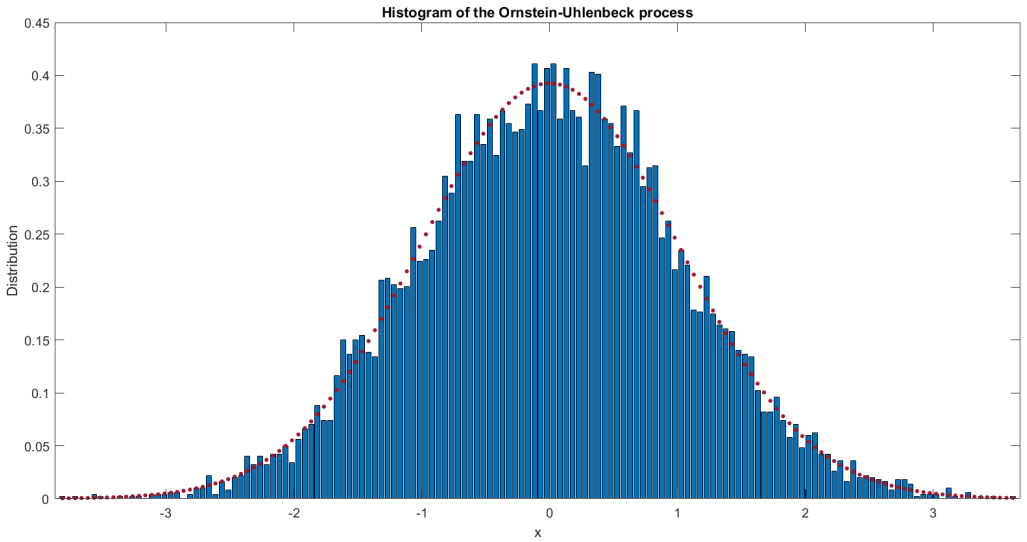


Figure C.28: Comparison between the histogram distribution trend and the Gaussian one with X_0=randn, T=10, N=500, n=10000, D=1, τ_{cor}=1.

Distribution of the Ornstein-Uhlenbeck Process with different n and T

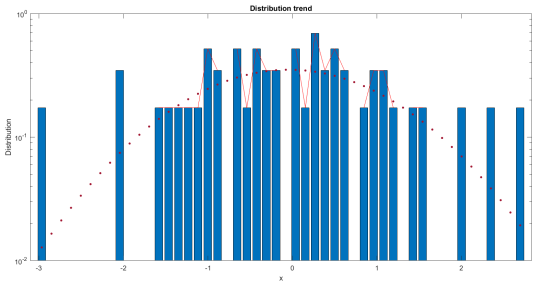
In order to study the distribution of the process and to compare it with the Gaussian one, we consider the empirical distribution of the histograms. Changing both the time T and the number of particles n it is possible to notice how the empirical distribution of the histogram resembles the Gaussian one as time and particles increase.

The *histograms_ou* MATLAB function performs a plot with the histogram, the empirical distribution and the Gaussian distribution.

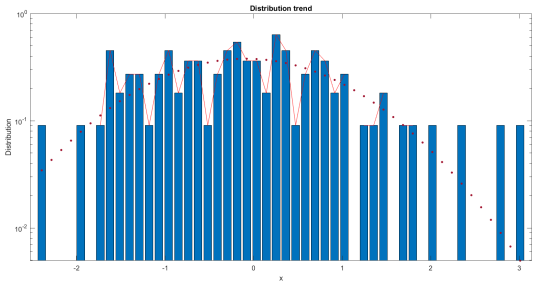
Changing the input arguments, it is possible to obtain different plots in order to highlight the behaviour of the Ornstein-Uhlenbeck process.

```
function [X]=histograms_ou(X0,T,N,n,D,tau)
% This function looks at the density distribution
% of the Ornstein-Uhlenbeck process and verifies
% that it has a Gaussian distribution changing times
% and the particle number.

%input: X0=initial point;
%       T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%       tau=correlation time;
%output:X= Ornstein-Uhlenbeck process;

%Ornstein-Uhlenbeck process construction
[X]=ornstein_uhlenbeck (X0,T,N,n,D,tau);

%histogram with empirical and gaussian distribution
[nn,y] = hist(X(:,end),50);
n_norm = (nn ./length(X(:,end))) ./(y(2)-y(1));
bar(y,n_norm); hold('on');
set(gca, 'YScale', 'log');
plot(y,n_norm,'r'); hold('on');
pd = fitdist(X(:,end),'Normal');
f = pdf(pd, y);
c=[0.6350 0.0780 0.1840];
scatter(y,f,10,c, 'filled')
```

```
28
   xlabel('x'), ylabel('Distribution')
30 title('Distribution trend')
   end
32
   %X0=randn; N=500; D=1; tau=1;
34 figure(1); n1=50; T1=50; [X]=histograms_ou(X0,T1,N,n1,D,tau);
   figure(2); n2=100; T2=100;[X]=histograms_ou(X0,T2,N,n2,D,tau);
36 figure(3); n3=250; T3=250; [X]=histograms_ou(X0,T3,N,n3,D,tau);
   figure(4); n4=500; T4=500; [X]=histograms_ou(X0,T4,N,n4,D,tau);
38 figure(5); n5=800; T5=800; [X]=histograms_ou(X0,T5,N,n5,D,tau);
   figure(6); n6=1000; T6=1000; [X]=histograms_ou(X0,T6,N,n6,D,tau);
```

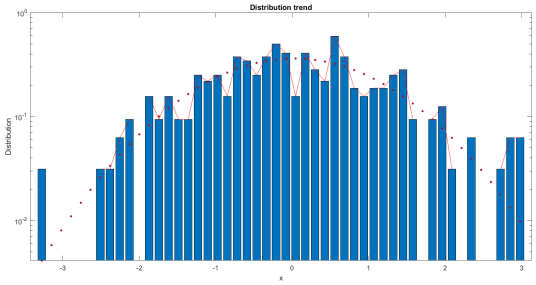


X0=randn, N=500, D=1, τ_{cor}=1, T=50, n=50.

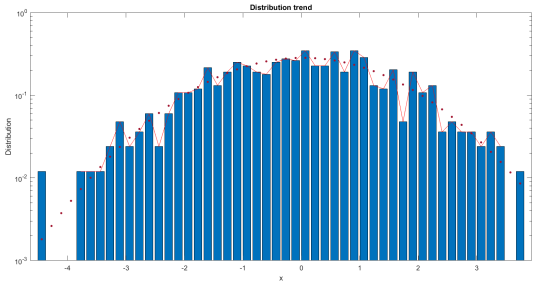


X0=randn, N=500, D=1, τ_{cor}=1, T=100, n=100.

Figure C.29

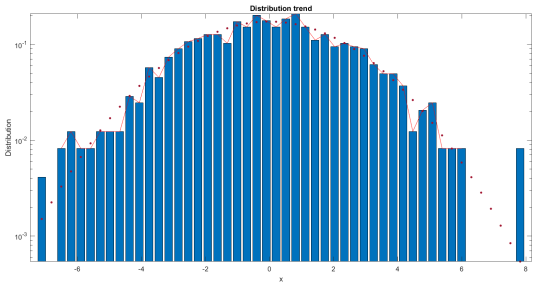


X0=randn, N=500, D=1, τ_{cor}=1, T=250, n=250.

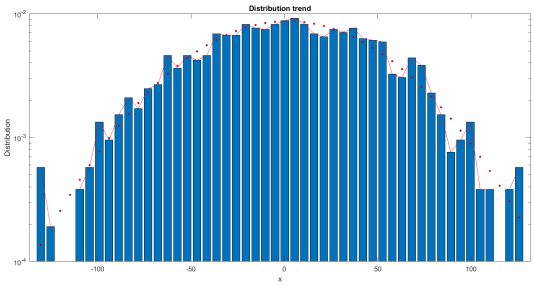


X0=randn, N=500, D=1, τ_{cor}=1, T=500, n=500.

Figure C.30



X0=randn, N=500, D=1, τ_{cor}=1, T=800, n=500.

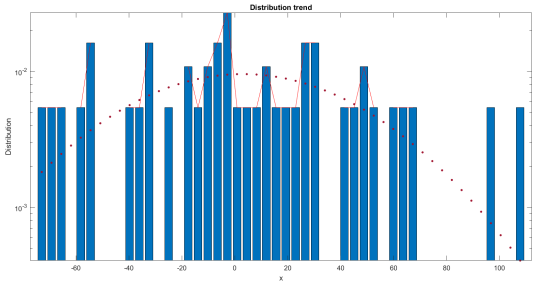


X0=randn, N=500, D=1, τ_{cor}=1, T=1000, n=1000.

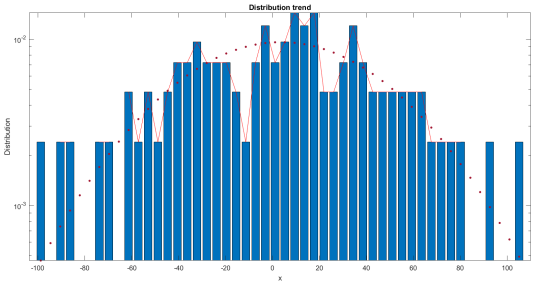
Figure C.31

If we fix the time T and we change the particles number n, it is possible to see how the fit gets better.

```
%X0=randn; T=1000; N=500; D=1; tau=1;

figure(1); n1=50; [X]=histograms_ou(X0,T,N,n1,D,tau);
figure(2); n2=100;[X]=histograms_ou(X0,T,N,n2,D,tau);
figure(3); n3=250; [X]=histograms_ou(X0,T,N,n3,D,tau);
figure(4); n4=500; [X]=histograms_ou(X0,T,N,n4,D,tau);
figure(5); n5=800; [X]=histograms_ou(X0,T,N,n5,D,tau);
figure(6); n6=1000; [X]=histograms_ou(X0,T,N,n6,D,tau);
```

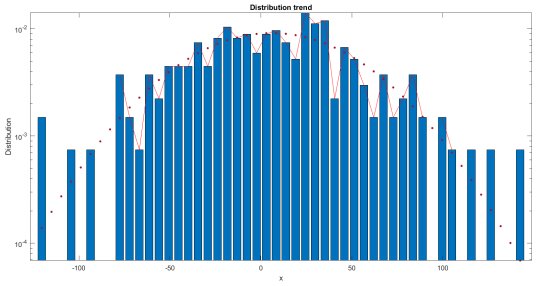


X0=randn, T=1000, N=500, D=1, τ_{cor}=1, n=50.

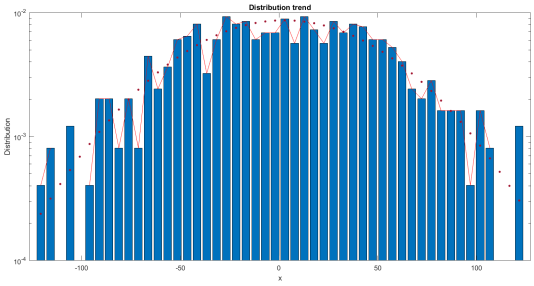


X0=randn, T=1000, N=500, D=1, τ_{cor}=1, n=100.

Figure C.32

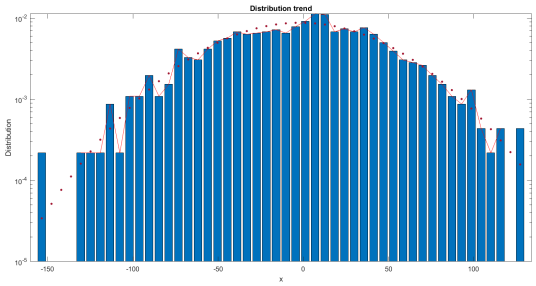


X0=randn, T=1000, N=500, D=1, τ_{cor}=1, n=250.

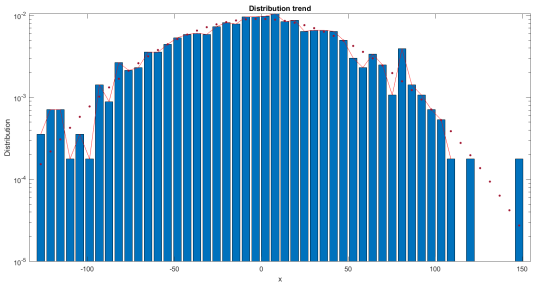


X0=randn, T=1000, N=500, D=1, τ_{cor}=1, n=500.

Figure C.33



X0=randn, T=1000, N=500, D=1, τ_{cor}=1, n=500.



X0=randn, T=1000, N=500, D=1, τ_{cor}=1, n=1000.

Figure C.34

C.3.4 Variance

The Ornstein–Uhlenbeck process is an example of a Gaussian process that has a bounded variance and admits a stationary probability distribution.

The variance trend is different form the linear one of the Wiener process, in fact for the The Ornstein–Uhlenbeck process the variance *saturate* after τ_{cor} that is the *correlation time*.

```
function [X,V]=variance_ou (X0,T,N,n,D,tau)
%This function displays the variance of the
%Ornstein-Uhlenbeck Process
%input: X0=initial point;
%       T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%       tau= correlation time;
%output:X= Ornstein-Uhlenbeck process;
%

%ornstein_uhlenbeck process
dt=T/N;
t=(0:dt:T); %instant of time vector
mu=0; %mean
sd=1; %standard deviation
h=sqrt(2.*D);
X=zeros(n,N);
X(:,1)=X0(:,1);
for i= 1:N
X(:,i+1)=X(:,i)-((X(:,i)./tau)*dt)+h*sqrt(dt)*normrnd(mu,sd,[n,1]);
end

V=var(X(:,:),1); %variance calculation with the matlab function 'var'
plot(t,V)
xlabel('Time'), ylabel('Variance')
title('Variance␣Trend')
end
```

If we fix all the inputs of the MATLAB function *variance_ou* and we only change the particles number n, it is possible to see how the trend gets better. This is showed in figures C.35, C.36, C.37.

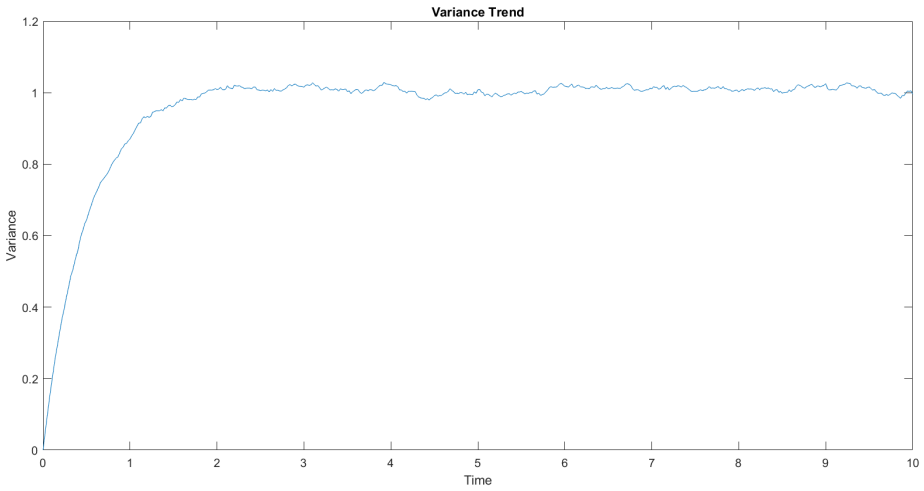


Figure C.35: X_0=randn, T=10, N=500, **n=10000**, D=1, τ_{cor}=1

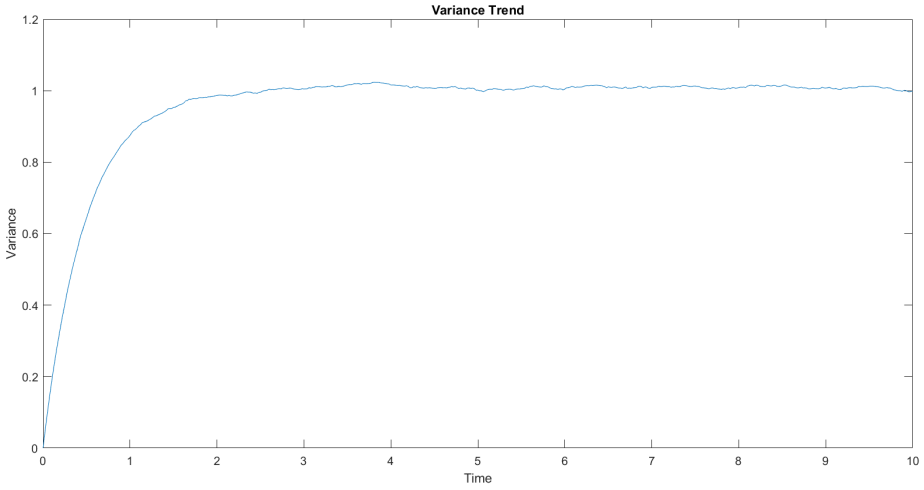


Figure C.36: X_0=randn, T=10, N=500, **n=80000**, D=1, τ_{cor}=1

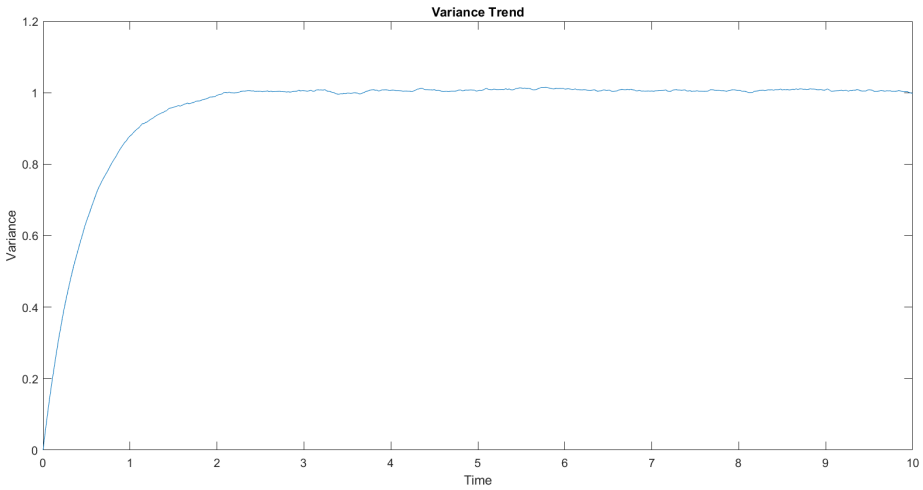


Figure C.37: X_0=randn, T=10, N=500, **n=100000**, D=1, τ_{cor}=1

Otherwise, if we fix the particles number $n = 10000$ and we change the correlation time τ, it is possible to notice that saturation rate changes, figure C.38.

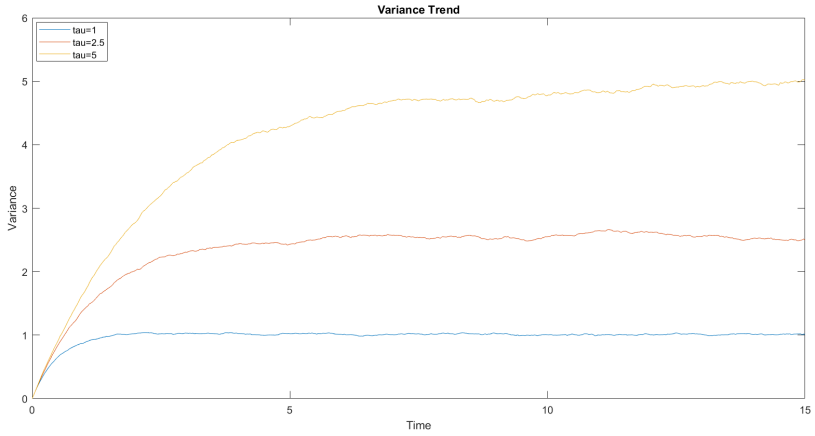


Figure C.38: X_0=randn, T=15, N=500, n=10000, D=1, τ_{cor}=**1**, τ_{cor}=**2.5**, τ_{cor}=**5**

Variance with respect to τ and D

In order to find the relationship between variance, correlation time and diffusion coefficient, we study before the variance behaviour with respect to τ and then with respect to D.

```
T=10; N=500; n=10000; D=1; X0=randn;
tau1=0.2; [~,V1]=variance_ou (X0,T,N,n,D,tau1); hold on;
tau2=0.4; [~,V2]=variance_ou (X0,T,N,n,D,tau2);
tau3=0.6; [~,V3]=variance_ou (X0,T,N,n,D,tau3);
tau4=0.8; [~,V4]=variance_ou (X0,T,N,n,D,tau4);
tau5=1; [~,V5]=variance_ou (X0,T,N,n,D,tau5);
tau6=1.2; [~,V6]=variance_ou (X0,T,N,n,D,tau6);
tau7=1.4; [~,V7]=variance_ou (X0,T,N,n,D,tau7);
tau8=1.6; [~,V8]=variance_ou (X0,T,N,n,D,tau8);
tau9=1.8; [~,V9]=variance_ou (X0,T,N,n,D,tau9);
tau10=2; [~,V10]=variance_ou (X0,T,N,n,D,tau10);
```

Using the function *variance_ou* the figure C.39 shows the variance behaviour as τ varies.

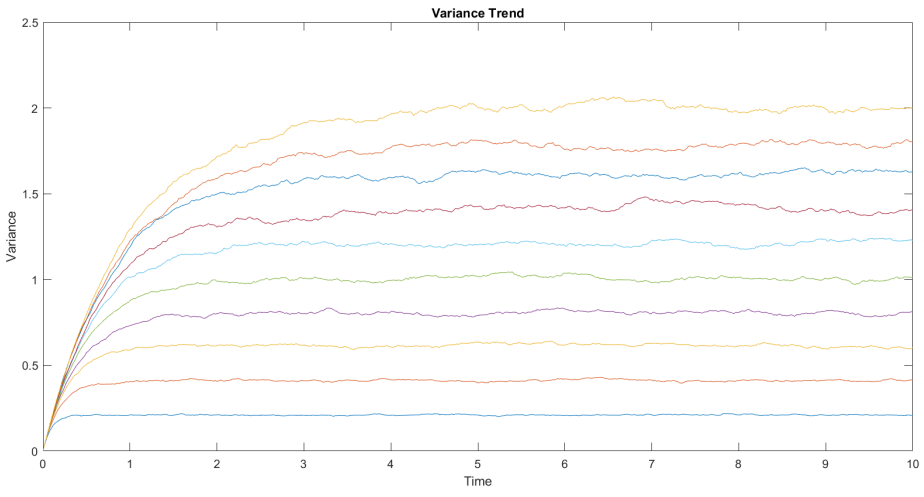


Figure C.39: X_0=randn, T=10, N=500, n=10000, D=1, τ_{cor}=**0.2**, τ_{cor}=**0.4**, τ_{cor}=**0.6**, τ_{cor}=**0.8**, τ_{cor}=**1**, τ_{cor}=**1.2**, τ_{cor}=**1.4**, τ_{cor}=**1.6**, τ_{cor}=**1.8**, τ_{cor}=**2**

Now we do the same for D and figure C.41 shows the variance behaviour as D varies.

```
T=10; N=500; n=10000; D=1; X0=randn;
D1=0.2; [~,V1]=variance_ou (X0,T,N,n,D1,tau); hold on;
D2=0.4; [~,V2]=variance_ou (X0,T,N,n,D2,tau);
D3=0.6; [~,V3]=variance_ou (X0,T,N,n,D3,tau);
D4=0.8; [~,V4]=variance_ou (X0,T,N,n,D4,tau);
D5=1; [~,V5]=variance_ou (X0,T,N,n,D5,tau);
D6=1.2; [~,V6]=variance_ou (X0,T,N,n,D6,tau);
D7=1.4; [~,V7]=variance_ou (X0,T,N,n,D7,tau);
D8=1.6; [~,V8]=variance_ou (X0,T,N,n,D8,tau);
D9=1.8; [~,V9]=variance_ou (X0,T,N,n,D9,tau);
D10=2; [~,V10]=variance_ou (X0,T,N,n,D10,tau);
```

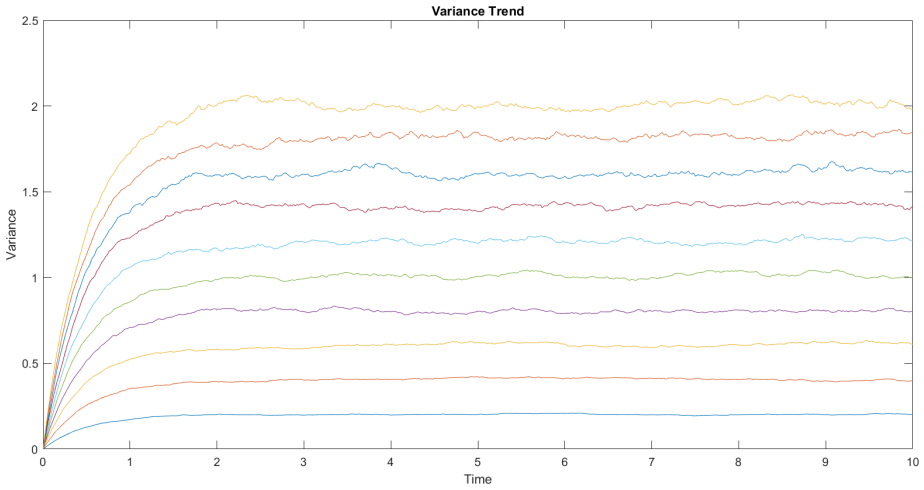


Figure C.40: X_0=randn, T=10, N=500, n=10000, τ_{cor}=1, **D=0.2**, **D=0.4**, **D=0.6**, **D=0.8**, **D=1**, **D=1.2**, **D=1.4**, **D=1.6**, **D=1.8**, **D=2**

We can now consider the two trends: (V, τ_{cor}) and (V, D).

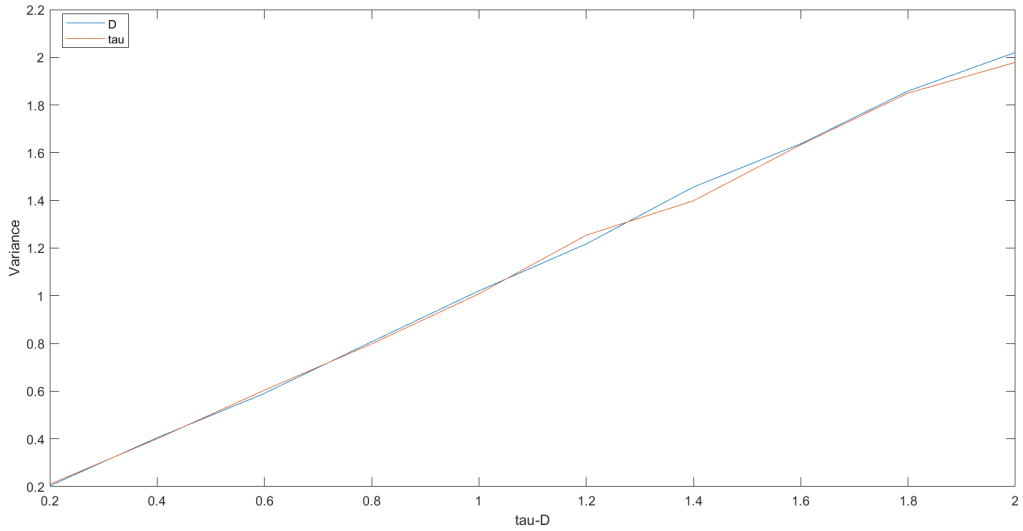


Figure C.41: X_0=randn, T=10, N=500, n=10000

C.3.5 Correlation

In Wiener process the correlation it was 0 because at each step there was the independence from the previous one.
In this case there is the dependence of each step from previous one so we want to verify that the correlation trajectory is exponential.

In order to compare the two trends, with the MATLAB function *correlation* we first calculate the correlation from the data, with the formula:

$$\mathbb{E}[X_t X_s] = \frac{1}{n-1} \sum_{i=1}^{N} \left(\frac{X_{t_i} - \bar{X}_t}{\sigma_t^2} \right) \left(\frac{X_{s_i} - \bar{X}_s}{\sigma_s^2} \right) \tag{C.14}$$

and then we calculate the analytical correlation with the formula:

$$C = \exp\left\{ -\frac{|t-s|}{\tau} \right\} \tag{C.15}$$

```
%X0=randn; T=10; N=500; n=10000; D=1; tau=1;
function [E,C] = correlation(X0,T,N,n,D,tau)
%input: X0=initial point;
%       T= time period;
%       N= discretization step;
%       n= particles number;
```

```
%        D= diffusion coefficient;
%        tau= correlation time;
%output:E= empirical correlation;
%        C= analytical correlation;

dt=T/N; %Discretization
t=(0:dt:T); %Vector of sample times
             %associated with all simulated paths

%Construction of the Ornstein-Uhlenbeck Process:
mu=0; %mean
sd=1; %standard deviation
h=sqrt(2.*D);
X=zeros(n,N);
X(:,1)=X0(:,1);
for i= 1:N
X(:,i+1)=X(:,i)-((X(:,i)./tau)*dt)+h*sqrt(dt)*normrnd(mu,sd,[n,1]);
end

%variance
V=var(X(:,:),1);

%empirical correlation
E=zeros(1,N+1);
m=mean(X);
for i=1:N+1
    for j=1:n
    E(i)=E(i)+(X(j,i)-m(i)).*(X(j,250)-m(250));
    end
    E(i)=E(i)./(V(i).*V(250));
    E(i)=E(i)./(n-1);
end
plot(t, E)

%analytic correlation
C=exp(-abs(t-5)./tau);
hold on
plot(t,C)

xlabel('Time'), ylabel('Correlation')
title('Correlation␣Trend')
end
```

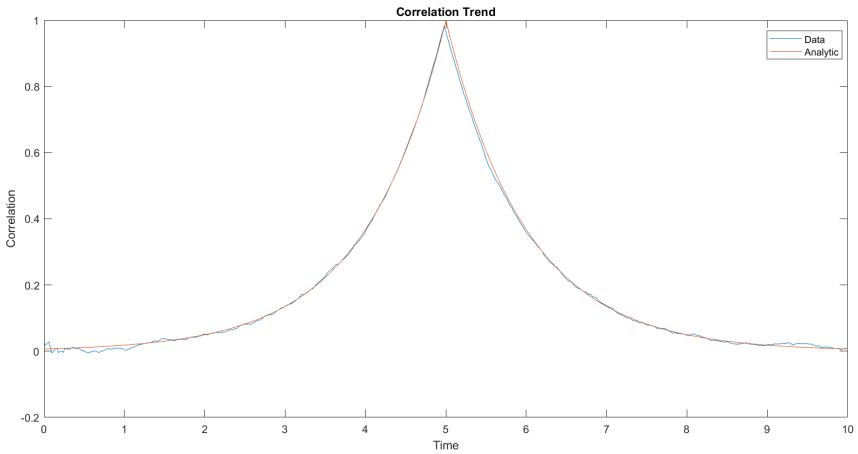


Figure C.42: X_0=randn, T=10, N=500, n=10000, D=1, τ_{cor}=1, s=5.

C.4 Langevin Equation

Langevin equation (LE) is a stochastic differential equation describing how a system evolves when subjected to a combination of deterministic and fluctuating ("random") forces.
LE contains both frictional and random forces. The dependent variables in a Langevin equation typically are collective (macroscopic) variables changing only slowly in comparison to the other (microscopic) variables of the system; the fluctuation-dissipation theorem relates the external driving force to the random internal force. The fast (microscopic) variables are responsible for the stochastic nature of the Langevin equation.

One application is to Brownian motion, which models the fluctuating motion of a small particle in a fluid.
The theory of Brownian motion has been extended to situations where the fluctuating object is not a real particle at all, but instead some collective porperty of a macroscopic system.

If we consider a colloidal system, we have three different timescales: τ_s, τ_B and τ_r. The first one is the *short atomic scale*: $\tau_s \approx 10^{-2}$ s; τ_B is the Brownian timescale for the relaxation of the particle velocity and $\tau_B \approx \frac{m}{\gamma} \approx 10^{-3}$ s; τ_r is the relaxation time for Brownian particle, i.e the time the particle have diffused its own radius: $\tau_r = \frac{a^2}{D}$.

In general

$$\tau_s << \tau_B << \tau_r \tag{C.16}$$

In classical mechanics we can describe particle motion with the Newton equation

$$m\frac{dv(t)}{dt} = F(t) \tag{C.17}$$

where $F(t)$ is the total instantaneous force on the particle at time t and it is due to the interaction of the Brownian particle with the surrounding medium.

In the case of a Brownian particle we expect a friction force and a random one. The friction force is given by $-\gamma v(t)$, with $\gamma = 6\pi\eta a$ and the random force $\xi(t)$ due to random density fluctuations in the fluid.
The equations of motion of the Brownian particle are:

$$\frac{dx(t)}{dt} = v(t) \tag{C.18}$$

$$\frac{dv(t)}{dt} = -\frac{\gamma}{m}v(t) + \frac{1}{m}\xi(t) \tag{C.19}$$

This is the *Langevin equations* of motion for the Brownian particle.

If we would neglect the random force $\xi(t)$, we obtain that the velocity of the Brownian particle is predicted to decay to zero at long times and this is in conflict with the equipartition theorem so the random force is therefore necessary to obtain the correct equilibrium.

In the conventional view of the fluctuation force it is supposed to come from occasional impacts of the Brownian particle with molecules of the surrounding medium. The force during an impact is supposed to vary extremely rapidly over the time of any observation and the effect of the fluctuating force can be summarized by giving its first and second moments

$$< \xi(t) >_\xi = 0, \quad < \xi(t_1)\xi(t_2) >_\xi = g\delta(t_1 - t_2) \tag{C.20}$$

The average is with respect to the distribution of the realizations of the stochastic variable $\xi(t)$. Since we have extracted the average force $-\gamma v(t)$ in the Langevin equation the average of the fluctuating force must by definition be zero, instead g is a measure of the strength of the fluctuation force.
The *delta* function in time indicates that there is no correlation between impacts in any distinct time intervals dt_1 and dt_2.

Another mathematical specification of this dynamical model is that the fluctuating force has a Gaussian distribution determined by the moments in (C.20). Those properties imply that $\xi(t)$ is a wildly fluctuating function, and it is not obvious that the differential equation (C.19) has a unique solution for a given initial condition, or even that $\frac{dv}{dt}$ exists. There is a standard existence theorem for differential equations which guarantee the existence of a local solution if $\xi(t)$ is continuous. A local solution is one which exists in some neighborhood of the point at which the initial value is given. But even if a solution exists it may be only local, or it may not be unique, unless some stronger conditions are imposed on $\xi(t)$.

An explicit formal solution of (C.19) is

$$v(t) = e^{-\frac{t}{\tau_B}} v(0) + \frac{1}{m} \int_0^t ds e^{-\frac{(t-s)}{\tau_B}} \xi(s) \tag{C.21}$$

but we don't know if this integral exists, so we write the (C.19) as

$$dv(t) = -\frac{\gamma}{m} v(t)dt + \frac{1}{m} dU(t) \tag{C.22}$$

where

$$dU(t) = \xi(t)dt \tag{C.23}$$

$U(t)$ is a continuous Markov process with zero mean. The continuity follows from (C.22) since

$$U(t) = U(0) + \int_0^t \xi(s)ds \tag{C.24}$$

and we must require that the integral be a continuous function of its upper limit, as for ordinary integrals.
On account of the randomness of the motion the random force $\xi(t)$ must average to zero, which is also implied by the separation in (C.19). If we choose our time origin so that $U(0) = 0$ we must have $< U(tk) >= 0$.

For what we said, the increments $U(t_1) - U(0), U(t_2) - U(t_1), ..., U(t_n) - U(t_{n-1})$ are independent. For long times we suppose that the random motion of the medium has attained a steady state. Then these increments are also stationary and identically distributed with zero mean.
Finally we can deduce that $U(t)$ is Gaussian with zero mean. Therefore, it

has all the requirements for a Wiener process, i.e.

$$U(t) = W(t) \tag{C.25}$$

so we can rewrite (C.22) as

$$dv(t) = -\frac{\gamma}{m}v(t)dt + \frac{1}{m}dW(t) \tag{C.26}$$

and the solution (C.21) becomes

$$v(t) = e^{-\frac{t}{\tau_B}}v(0) + \frac{1}{m}\int_0^t dse^{-\frac{(t-s)}{\tau_B}}dW(s) \tag{C.27}$$

C.4.1 Computational view

Considered an Ornstein-Uhlenbeck process V, we have

$$dV = -\frac{V}{\tau}dt + \sqrt{2D}dW \tag{C.28}$$

and we want find the process X as

$$dX = Vdt \tag{C.29}$$

Considering a time interval $[0, T]$, N discretization step and n number of particles, the MATLAB function *langevin* performs the Langevin Process corresponding to each considered particle.
Here the diffusion coefficient D and the correlation time τ are chosen equal to 1 and we use the MATLAB function normrnd to generate a random numbers from the normal distribution with mean $\mu = 0$ and standard deviation $sd = 1$ for the construction of the Ornstein-Uhlenbeck process. For this one the initial point X_0 is random instead for the Langevin process is equal to 0.

```
%X0=randn; T=50; N=100; n=10000; D=1; tau=1
function [X]=langevin(X0,T,N,n,D,tau)
%This function creates the Langevin Process
%input: X0=initial point;
%       T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%       tau=correlation time;
%output:X= Langevin process;
```

```
dt=T/N;
t=(0:dt:T); %instant of time vector

%Ornstein-Uhlenbeck process
mu=0; %mean
sd=1; %standard deviation
h=sqrt(2*D);
v=zeros(n,N);
v(:,1)=X0(:,1);
%normrnd: generates a random number from the normal
%          distribution with mean parameter mu and
%          standard deviation parameter sd [N,1]
%          indicates the size of each dimension
for i= 1:N
v(:,i+1)=v(:,i)-((v(:,i)/tau)*dt)+h*sqrt(dt)*normrnd(mu,sd,[n,1]);
end
%costruction of Langevin process
X=zeros(n,N); %initial point= 0
for i= 1:N
    X(:,i+1)=X(:,i)+v(:,i)*dt;
end
plot(t,X)
xlabel('Time'), ylabel('Process State')
title('Langevin Equation Application')
end
```

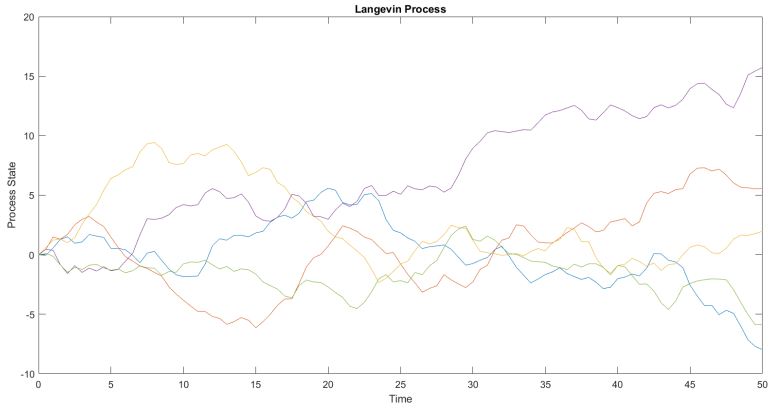


Figure C.43: X_0=randn, T=50, N=100, n=5, D=1, τ_{cor}=1

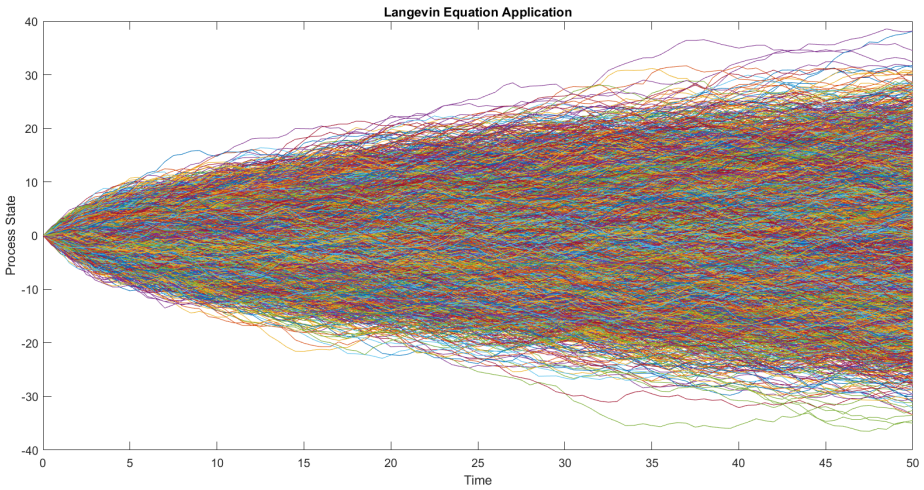


Figure C.44: X_0=randn, T=50, N=100, n=10000, D=1, τ_{cor}=1

C.4.2 Distribution of Langevin Process

In this section we want to study the distribution of the Langevin process and verify if it can be compared with a Gaussian distribution. As we did for the previous processes, in order to do this comparison we use the *hist_tot_lang* MATLAB function which first create the Langevin Process for $n = 10000$ particles and then the corresponding histogram.

The trend of the histogram was then considered and subsequently it has been compared with the Gaussian trend both with logarithmic scale and linear scale.

```
function [X]=hist_tot_lang(X0,T,N,n,D,tau)
% This function looks at the density distribution of the Langevin
% process and verifies that it has a Gaussian distribution.

%input: X0=initial point;
%       T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%       tau=correlation time;
%output:X=Langevin process;

%Langevin process construction
```

```
14 [X]=langevin(X0,T,N,n,D,tau);

16 figure(1)
   n_bins=150;
18 [nn,y] = hist(X(:,end),n_bins); % hist matlab function
   n_norm = (nn ./length(X(:,end))) ./(y(2)-y(1));
20                                  % normalize hist function
   bar(y,n_norm); hold ('on');
22 plot(y,n_norm,'r');
   xlabel('x'), ylabel('Distribution')
24 title('')

26 figure(2);
   semilogy(y,n_norm,'r'); hold('on');
28 pd = fitdist(X(:,end),'Normal'); %creating the gaussian function
   f = pdf(pd, y);
30 c1=[0.6350 0.0780 0.1840];
   scatter(y,f,10,c1, 'filled')
32 xlabel('x'), ylabel('Distribution')
   title('')
34
   figure(3)
36 bar(y,n_norm); hold('on');
   pd = fitdist(X(:,end),'Normal');
38 f = pdf(pd, y);
   c=[0 0.4470 0.7410];
40 scatter(y,f,10,c1, 'filled')
   set(gca,'yscale','log')
42 xlabel('x'), ylabel('Distribution')
   title('')
44
   figure(4)
46 plot(y,n_norm,'r'); hold('on');
   pd = fitdist(X(:,end),'Normal'); %creating the gaussian function
48 f = pdf(pd, y);
   c1=[0.6350 0.0780 0.1840];
50 scatter(y,f,10,c1, 'filled')
   xlabel('x'), ylabel('Distribution')
52 title('')

54 figure(5)
   bar(y,n_norm); hold('on');
56 pd = fitdist(X(:,end),'Normal');
```

```
f = pdf(pd, y);
c=[0 0.4470 0.7410];
scatter(y,f,10,c1, 'filled')
xlabel('x'), ylabel('Distribution')
title('')

end
```

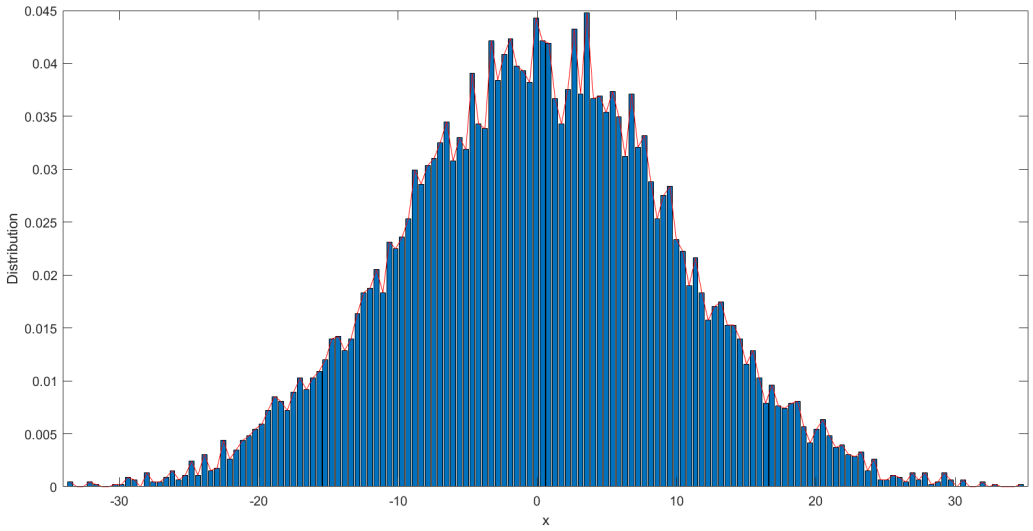


Figure C.45: Histogram of $X(t)$ and its distribution with X_0=randn, T=50, N=100, n=10000, D=1, τ_{cor}=1.

Distribution of Langevin Process changing the number of particles

As we did for the previous processes, in order to study the distribution of the process and to compare it with the Gaussian one, we consider the empirical distribution of the histograms.

Changing the number of particles n it is possible to notice how the empirical distribution of the histogram resembles the Gaussian one as time and particles increase.

The *histograms_lang* MATLAB function performs a plot with the histogram, the empirical distribution and the Gaussian distribution.

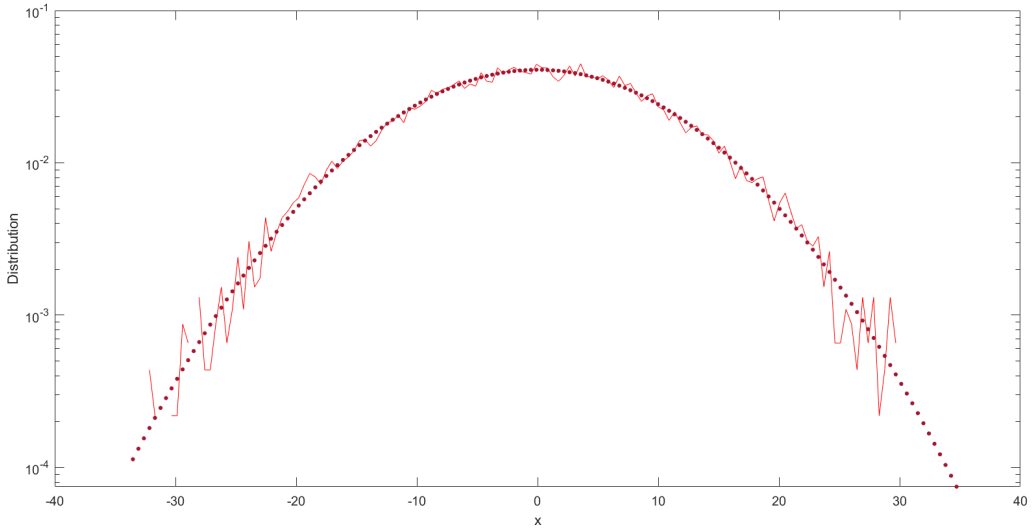


Figure C.46: Comparison between the histogram distribution trend and the Gaussian one with X_0=randn, T=50, N=100, n=10000, D=1, τ_{cor}=1.

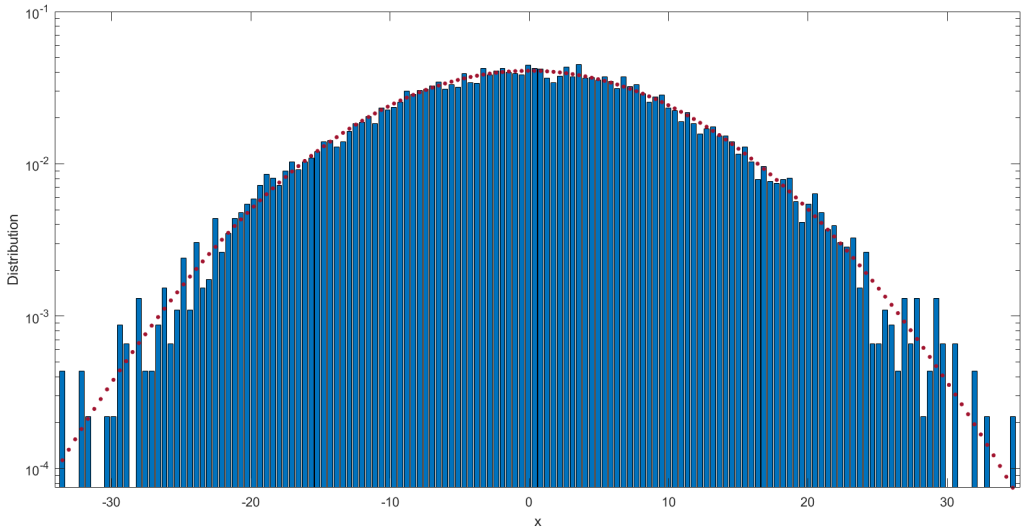


Figure C.47: Gaussian distribution on $X(t)$ histogram withX_0=randn, T=50, N=100, n=10000, D=1, τ_{cor}=1

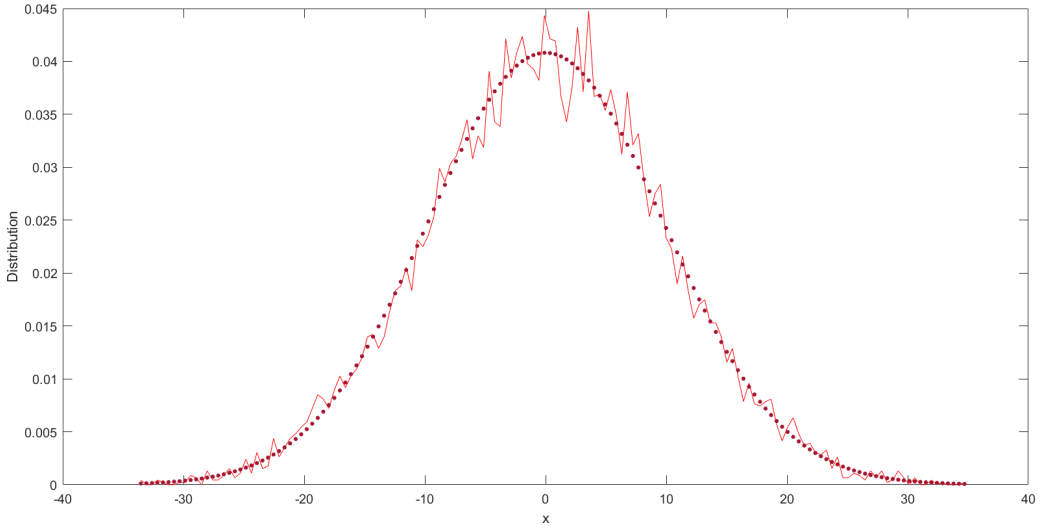


Figure C.48: Histogram of $X(t)$ and its distribution with X_0=randn, T=50, N=100, n=10000, D=1, τ_{cor}=1

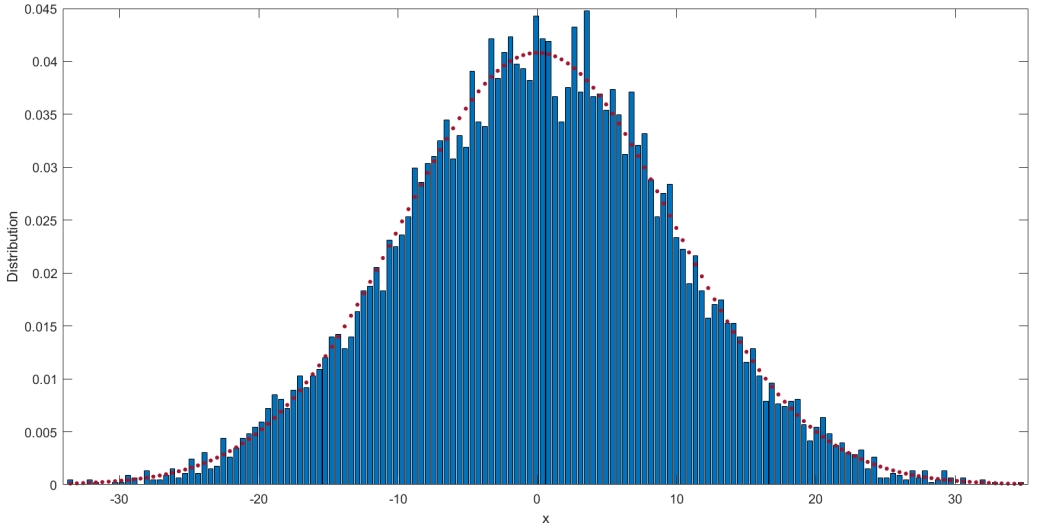
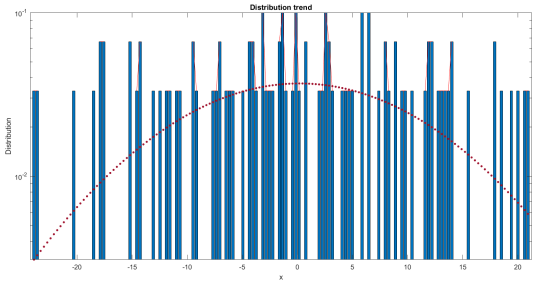
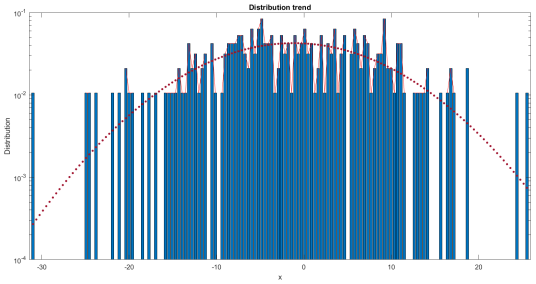


Figure C.49: Comparison between the histogram distribution trend and the Gaussian one with X_0=randn, T=50, N=100, n=10000, D=1, τ_{cor}=1

```
function [X]=histograms_lang(X0,T,N,n,D,tau)
% This function looks at the density distribution of the
% Langevin process and verifies that it has a
% Gaussian distribution changing times
% and the particle number.

%input: X0=initial point;
%       T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%       tau=correlation time;
%output:X= Langevin process;

%Langevin process construction
[X]=langevin(X0,T,N,n,D,tau);

%histogram with empirical and gaussian distribution
[nn,y] = hist(X(:,end),150);
n_norm = (nn ./length(X(:,end))) ./(y(2)-y(1));
bar(y,n_norm); hold('on');
set(gca, 'YScale', 'log');
plot(y,n_norm,'r'); hold('on');
pd = fitdist(X(:,end),'Normal');
f = pdf(pd, y);
c=[0.6350 0.0780 0.1840];
scatter(y,f,10,c, 'filled')

xlabel('x'), ylabel('Distribution')
title('Distribution trend')
end

%X0=randn; T=50; N=100; D=1; tau=1;
figure(1); n1=100; [X]=histograms_lang(X0,T,N,n1,D,tau);
figure(2); n2=250;[X]=histograms_lang(X0,T,N,n2,D,tau);
figure(3); n3=500; [X]=histograms_lang(X0,T,N,n3,D,tau);
figure(4); n4=800; [X]=histograms_lang(X0,T,N,n4,D,tau);
figure(5); n5=1000; [X]=histograms_lang(X0,T,N,n5,D,tau);
figure(6); n6=5000; [X]=histograms_lang(X0,T,N,n6,D,tau);
```

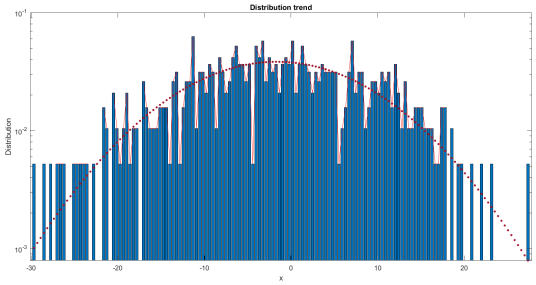


X0=randn, T=50, N=100, D=1, τ_{cor}=1, n=100.

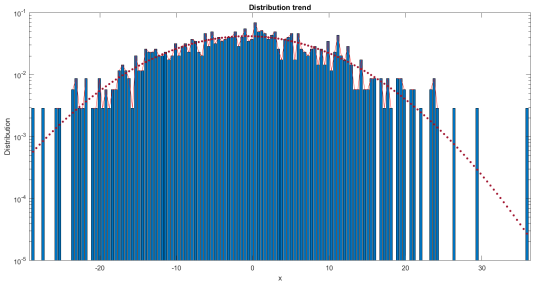


X0=randn, T=50, N=100, D=1, τ_{cor}=1, n=250.

Figure C.50

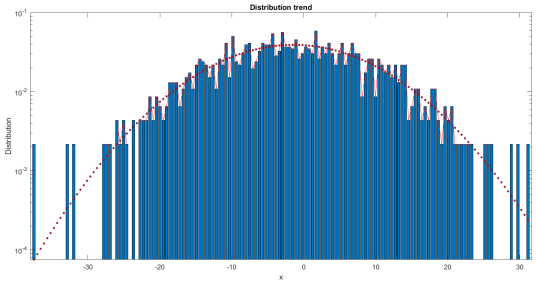


X0=randn, T=50, N=100, D=1, τ_{cor}=1, n=500.

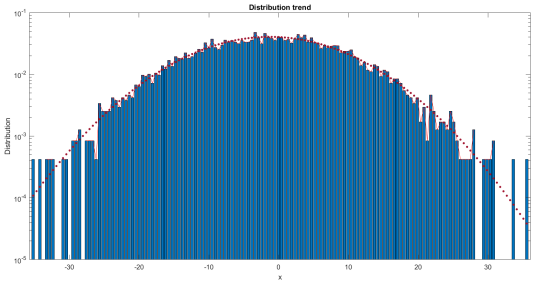


X0=randn, T=50, N=100, D=1, τ_{cor}=1, n=800.

Figure C.51



X0=randn, T=50, N=100, D=1, τ_{cor}=1, n=1000.



X0=randn, T=50, N=100, D=1, τ_{cor}=1, n=1000.

Figure C.52

C.4.3 Variance

The aim of this section is to show the variance behaviour with respect to time: we want to calculate the variance σ^2 and show that its trend before increase asymptotically as t^2 and then linear with respect to t.

With the MATLAB function *variance_lang* we calculate the variance of the process $X(t)$ with the function *var* and then it is plotted and it is also compared with the Ornstein-Uhlenbeck process variance.

```
%X0=randn; T=10; N=100; n=10000; D=1; tau=1;
function [V,Vo]=variance_lang (X0,T,N,n,D,tau)
%This function displays the variance of the Langevin Process
%input: X0=initial point;
%       T= time period;
%       N= discrtizion step;
%       n= particles number;
%       D= diffusion coefficient;
%       tau= correlation time;
%output:V=variance of the Langevin process;
%       Vo=variance of the Ornstein-Uhlenbeck;
%
[X,v]=langevin(X0,T,N,n,D,tau);

dt=T/N;
t=(0:dt:T); %instant of time vector
Vo=var(v(:,:),1); %variance of the Ornstein-Uhlenbeck
V=var(X(:,:),1); %variance calculation with the matlab function 'var'
figure(2);
plot(t,V); hold on;
plot(t,Vo)
xlabel('Time'), ylabel('Variance')
title('Variance Trend')
end
```

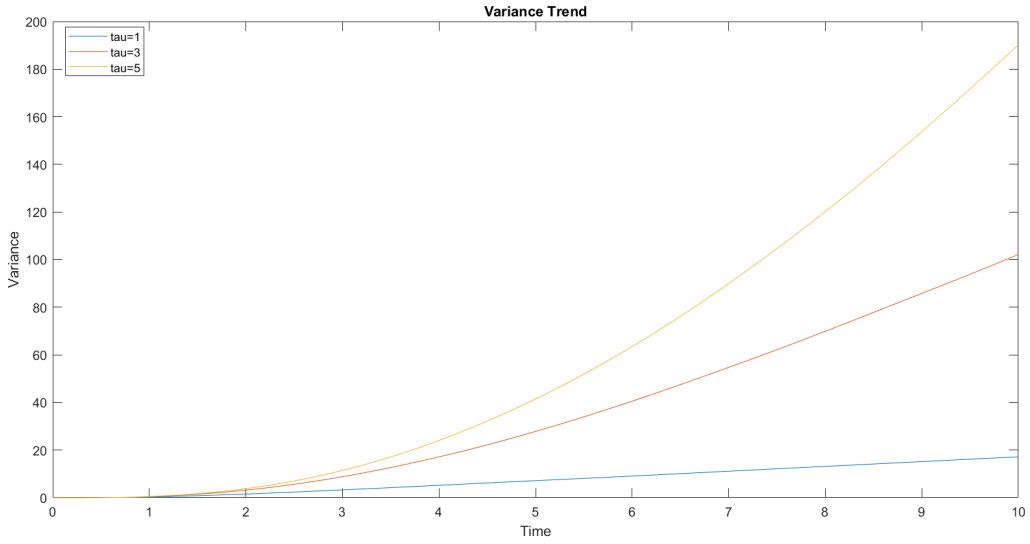


Figure C.53: X_0=randn, T=10, N=100, n=10000, D=1, τ_{cor}=1, τ_{cor}=3, τ_{cor}=5

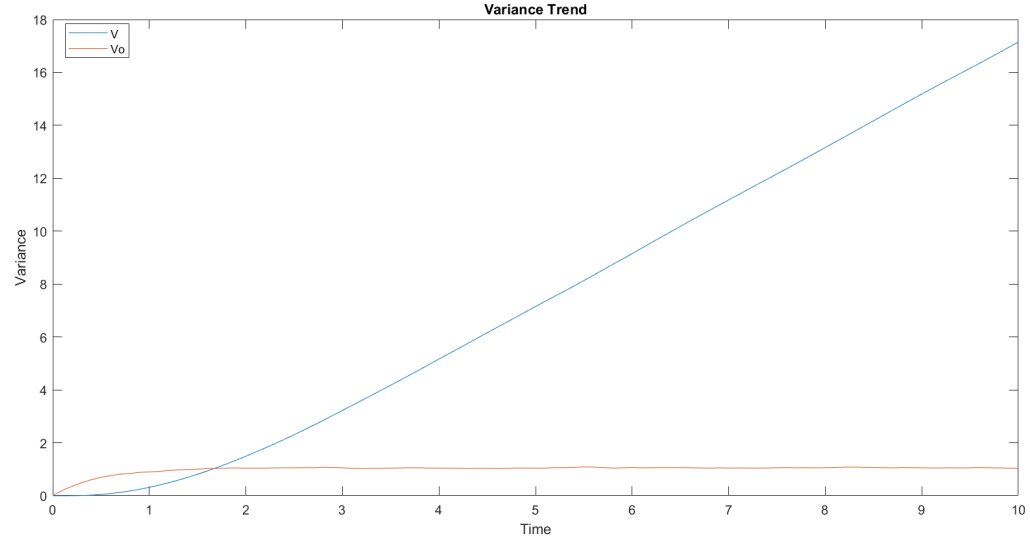


Figure C.54: X_0=randn, T=10, N=100, n=10000, D=1, τ_{cor}=1

In the following figure is showed the comparison between the variance from data and

$$Va = 2\ D\ \tau^2\ t \tag{C.30}$$

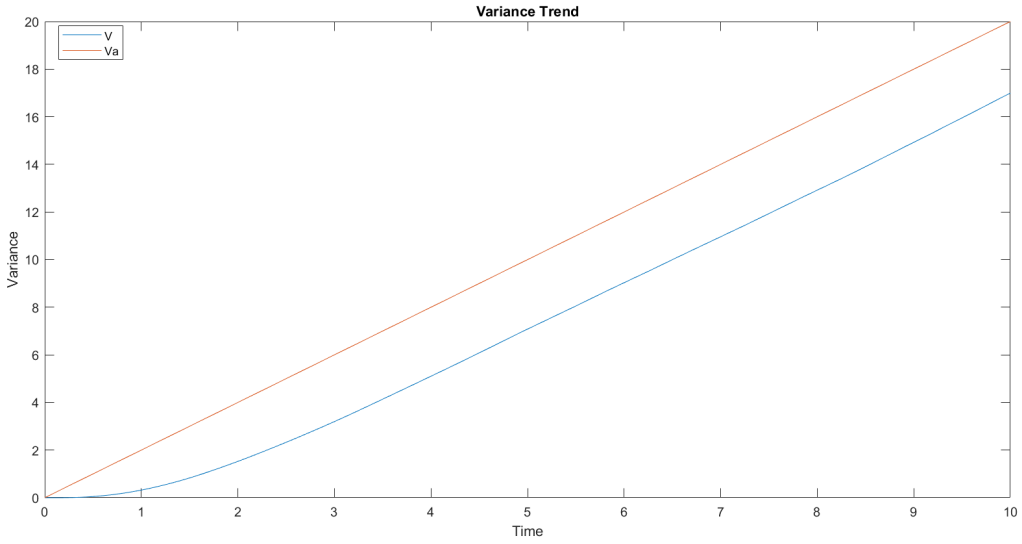


Figure C.55: X_0=randn, T=10, N=100, n=10000, D=1, τ_{cor}=1

Bibliography

[1] Alexandru Agapie et al. “Predictability in cellular automata”. In: *Plos one* 9.10 (2014), e108177.

[2] A Alexandridis et al. “Wildland fire spread modelling using cellular automata: evolution in large-scale spatially heterogeneous environments under fire suppression tactics”. In: *International Journal of Wildland Fire* 20.5 (2011), pp. 633–647.

[3] Alex Alexandridis et al. “A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990”. In: *Applied Mathematics and Computation* 204.1 (2008), pp. 191–201.

[4] EA Alhassan, L Albert, and O Louis. “On some Applications of Beta Function in some Statistical Distributions”. In: ().

[5] Frédéric Allaire, Jean-Baptiste Filippi, and Vivien Mallet. “Generation and evaluation of an ensemble of wildland fire simulations”. In: *International journal of wildland fire* 29.2 (2020), pp. 160–173.

[6] Frédéric Allaire, Vivien Mallet, and Jean-Baptiste Filippi. “Novel method for a posteriori uncertainty quantification in wildland fire spread simulation”. In: *Applied Mathematical Modelling* 90 (2021), pp. 527–546.

[7] GE Andreus, R Askey, and Ranjan Roy. *Special Functions, Encyclopedia of Mathematics and Its Applications, Vol. 71*. 1999.

[8] Paolo Baldi. “Stochastic calculus”. In: *Stochastic Calculus*. Springer, 2017, pp. 215–254.

[9] A Collin, D Bernardin, and O Sero-Guillaume. “A physical-based cellular automaton model for forest-fire propagation”. In: *Combustion science and technology* 183.4 (2011), pp. 347–369.

[10] Vera N Egorova, Andrea Trucchia, and Gianni Pagnini. “Fire-spotting generated fires. Part I: The role of atmospheric stability”. In: *Applied Mathematical Modelling* 84 (2020), pp. 590–609.

[11] Vera N Egorova, Andrea Trucchia, and Gianni Pagnini. "Fire-spotting generated fires. Part II: The role of flame geometry and slope". In: *Applied Mathematical Modelling* 104 (2022), pp. 1–20.

[12] Orin J Farrell and Bertram Ross. *Solved problems in analysis: as applied to gamma, beta, legendre and bessel functions*. Courier Corporation, 2013.

[13] Niloy Ganguly et al. "A survey on cellular automata". In: (2003).

[14] Tiziano Ghisu et al. "An optimal Cellular Automata algorithm for simulating wildfire spread". In: *Environmental Modelling & Software* 71 (2015), pp. 1–14.

[15] Leon M Hall. "Special functions". In: (1995).

[16] Desmond J Higham. "An algorithmic introduction to numerical simulation of stochastic differential equations". In: *SIAM review* 43.3 (2001), pp. 525–546.

[17] Samuel Kotz and Johan René Van Dorp. *Beyond beta: other continuous families of distributions with bounded support and applications*. World Scientific, 2004.

[18] Nikola Nikolaevich Lebedev, Richard A Silverman, and DB Livhtenberg. "Special functions and their applications". In: *Physics Today* 18.12 (1965), p. 70.

[19] Marcos López-De-Castro et al. "Physical and Non-Physical Fire-Spotting Models: A Comparison Study by a Wildfire Simulator Based on a Cellular Automata Approach". In: *Environmental Sciences Proceedings* 17.1 (2022), p. 27.

[20] Saralees Nadarajah and Samuel Kotz. "Multitude of beta distributions with applications". In: *Statistics* 41.2 (2007), pp. 153–179.

[21] Grigorios A Pavliotis. *Stochastic processes and applications: diffusion processes, the Fokker-Planck and Langevin equations*. Vol. 60. Springer, 2014.

[22] Holly A Perryman et al. "A cellular automata model to link surface fires to firebrand lift-off and dispersal". In: *International journal of wildland fire* 22.4 (2012), pp. 428–439.

[23] D Riddhi. "Beta function and its applications". In: *The University of Tennesse, Knoxville, USA.[online] Available from: http://sces. phys. utk. edu/moreo/mm08/Riddi. pdf* (2008).

[24] Ian N Sneddon and E Richard Cohen. "Special functions of mathematical physics and chemistry". In: *Physics Today* 9.11 (1956), p. 46.

[25] Andrea Trucchia et al. “RandomFront 2.3: a physical parameterisation of fire spotting for operational fire spread models–implementation in WRF-SFIRE and response analysis with LSFire+”. In: *Geoscientific Model Development* 12.1 (2019), pp. 69–87.

[26] Andrea Trucchia et al. “PROPAGATOR: an operational cellular-automata based wildfire simulator”. In: *Fire* 3.3 (2020), p. 26.

[27] Domingos Xavier Viegas. “Forest fire propagation”. In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 356.1748 (1998), pp. 2907–2928.